

1. *Phragmites australis* (Cav.) Trin. ex Steud.
 2. *Spartina patens* (Muhl.) Bosc.
 3. *Spartina alterniflora* (Lois.) Bosc.
 4. *Spartina cynosuroides* (L.) Bosc.
 5. *Spartina anglica* (Muhl.) Bosc.
 6. *Spartina pectinata* (L.) Bosc.
 7. *Spartina rigida* (L.) Bosc.
 8. *Spartina foliosa* (Muhl.) Bosc.
 9. *Spartina rostrata* (L.) Bosc.
 10. *Spartina angustata* (L.) Bosc.
 11. *Spartina gracilis* (L.) Bosc.
 12. *Spartina tenuifolia* (L.) Bosc.
 13. *Spartina robusta* (L.) Bosc.
 14. *Spartina densa* (L.) Bosc.
 15. *Spartina serotina* (L.) Bosc.
 16. *Spartina angustata* (L.) Bosc.
 17. *Spartina gracilis* (L.) Bosc.
 18. *Spartina tenuifolia* (L.) Bosc.
 19. *Spartina robusta* (L.) Bosc.
 20. *Spartina densa* (L.) Bosc.
 21. *Spartina serotina* (L.) Bosc.
 22. *Spartina angustata* (L.) Bosc.
 23. *Spartina gracilis* (L.) Bosc.
 24. *Spartina tenuifolia* (L.) Bosc.
 25. *Spartina robusta* (L.) Bosc.
 26. *Spartina densa* (L.) Bosc.
 27. *Spartina serotina* (L.) Bosc.
 28. *Spartina angustata* (L.) Bosc.
 29. *Spartina gracilis* (L.) Bosc.
 30. *Spartina tenuifolia* (L.) Bosc.
 31. *Spartina robusta* (L.) Bosc.
 32. *Spartina densa* (L.) Bosc.
 33. *Spartina serotina* (L.) Bosc.
 34. *Spartina angustata* (L.) Bosc.
 35. *Spartina gracilis* (L.) Bosc.
 36. *Spartina tenuifolia* (L.) Bosc.
 37. *Spartina robusta* (L.) Bosc.
 38. *Spartina densa* (L.) Bosc.
 39. *Spartina serotina* (L.) Bosc.
 40. *Spartina angustata* (L.) Bosc.
 41. *Spartina gracilis* (L.) Bosc.
 42. *Spartina tenuifolia* (L.) Bosc.
 43. *Spartina robusta* (L.) Bosc.
 44. *Spartina densa* (L.) Bosc.
 45. *Spartina serotina* (L.) Bosc.
 46. *Spartina angustata* (L.) Bosc.
 47. *Spartina gracilis* (L.) Bosc.
 48. *Spartina tenuifolia* (L.) Bosc.
 49. *Spartina robusta* (L.) Bosc.
 50. *Spartina densa* (L.) Bosc.
 51. *Spartina serotina* (L.) Bosc.
 52. *Spartina angustata* (L.) Bosc.
 53. *Spartina gracilis* (L.) Bosc.
 54. *Spartina tenuifolia* (L.) Bosc.
 55. *Spartina robusta* (L.) Bosc.
 56. *Spartina densa* (L.) Bosc.
 57. *Spartina serotina* (L.) Bosc.
 58. *Spartina angustata* (L.) Bosc.
 59. *Spartina gracilis* (L.) Bosc.
 60. *Spartina tenuifolia* (L.) Bosc.
 61. *Spartina robusta* (L.) Bosc.
 62. *Spartina densa* (L.) Bosc.
 63. *Spartina serotina* (L.) Bosc.
 64. *Spartina angustata* (L.) Bosc.
 65. *Spartina gracilis* (L.) Bosc.
 66. *Spartina tenuifolia* (L.) Bosc.
 67. *Spartina robusta* (L.) Bosc.
 68. *Spartina densa* (L.) Bosc.
 69. *Spartina serotina* (L.) Bosc.
 70. *Spartina angustata* (L.) Bosc.
 71. *Spartina gracilis* (L.) Bosc.
 72. *Spartina tenuifolia* (L.) Bosc.
 73. *Spartina robusta* (L.) Bosc.
 74. *Spartina densa* (L.) Bosc.
 75. *Spartina serotina* (L.) Bosc.
 76. *Spartina angustata* (L.) Bosc.
 77. *Spartina gracilis* (L.) Bosc.
 78. *Spartina tenuifolia* (L.) Bosc.
 79. *Spartina robusta* (L.) Bosc.
 80. *Spartina densa* (L.) Bosc.
 81. *Spartina serotina* (L.) Bosc.
 82. *Spartina angustata* (L.) Bosc.
 83. *Spartina gracilis* (L.) Bosc.
 84. *Spartina tenuifolia* (L.) Bosc.
 85. *Spartina robusta* (L.) Bosc.
 86. *Spartina densa* (L.) Bosc.
 87. *Spartina serotina* (L.) Bosc.
 88. *Spartina angustata* (L.) Bosc.
 89. *Spartina gracilis* (L.) Bosc.
 90. *Spartina tenuifolia* (L.) Bosc.
 91. *Spartina robusta* (L.) Bosc.
 92. *Spartina densa* (L.) Bosc.
 93. *Spartina serotina* (L.) Bosc.
 94. *Spartina angustata* (L.) Bosc.
 95. *Spartina gracilis* (L.) Bosc.
 96. *Spartina tenuifolia* (L.) Bosc.
 97. *Spartina robusta* (L.) Bosc.
 98. *Spartina densa* (L.) Bosc.
 99. *Spartina serotina* (L.) Bosc.
 100. *Spartina angustata* (L.) Bosc.

5

10

15

20

25

30

35

5

30

[illegible]

United States Patent No. 4,922,341 discloses a method for scene-model-assisted reduction of image data for digital television signals, whereby a picture signal supplied at time is to be coded, whereby a predecessor frame from a scene
5 already coded at time $t-1$ is present in an image store as a reference, and whereby the frame-to-frame information is composed of an amplification factor, a shift factor, and an adaptively acquired quad-tree division structure. Upon initialization of the system, a uniform, prescribed gray
10 scale value or picture half-tone expressed as a defined luminance value is written into the image store of a coder at the transmitter and in the image store of a decoder at the receiver store, in the same way for all picture elements (pixels). Both the image store in the coder as well as the
15 image store in the decoder are each operated with feed back to themselves in a manner such that the content of the image store in the coder and decoder can be read out in blocks of variable size, can be amplified with a factor greater than or less than 1 of the luminance and can be written back into the
20 image store with shifted addresses, whereby the blocks of variable size are organized according to a known quad tree data structure.

United States Patent No. 5,122,875 discloses an apparatus for encoding/decoding an HDTV signal. The
25 apparatus includes a compression circuit responsive to high definition video source signals for providing hierarchically layered codewords CW representing compressed video data and associated codewords T, defining the types of data represented by the codewords CW. A priority selection
30 circuit, responsive to the codewords CW and T, parses the codewords CW into high and low priority codeword sequences wherein the high and low priority codeword sequences correspond to compressed video data of relatively greater and lesser importance to image reproduction respectively. A

transport processor, responsive to the high and low priority codeword sequences, forms high and low priority transport blocks of high and low priority codewords, respectively. Each transport block includes a header, codewords CW and error detection check bits. The respective transport blocks are applied to a forward error check circuit for applying additional error check data.. Thereafter, the high and low priority data are applied to a modem wherein quadrature amplitude modulates respective carriers for transmission.

10 United States Patent No. 5,146,325 discloses a video decompression system for decompressing compressed image data wherein odd and even fields of the video signal are independently compressed in sequences of intraframe and interframe compression modes and then interleaved for transmission. The odd and even fields are independently decompressed. During intervals when valid decompressed odd/even field data is not available, even/odd field data is substituted for the unavailable odd/even field data. Independently decompressing the even and odd fields of data and substituting the opposite field of data for unavailable data may be used to advantage to reduce image display latency during system start-up and channel changes.

25 United States Patent No. 5,168,356 discloses a video signal encoding system that includes apparatus for segmenting encoded video data into transport blocks for signal transmission. The transport block format enhances signal recovery at the receiver by virtue of providing header data from which a receiver can determine re-entry points into the data stream on the occurrence of a loss or corruption of transmitted data. The re-entry points are maximized by providing secondary transport headers embedded within encoded video data in respective transport blocks.

30 United States Patent No. 5,168,375 discloses a method for processing a field of image data samples to provide for

FOUO 99999999

one or more of the functions of decimation, interpolation, and sharpening. This is accomplished by an array transform processor such as that employed in a JPEG compression system. Blocks of data samples are transformed by the discrete even cosine transform (DECT) in both the decimation and interpolation processes, after which the number of frequency terms is altered. In the case of decimation, the number of frequency terms is reduced, this being followed by inverse transformation to produce a reduced-size matrix of sample points representing the original block of data. In the case of interpolation, additional frequency components of zero value are inserted into the array of frequency components after which inverse transformation produces an enlarged data sampling set without an increase in spectral bandwidth. In the case of sharpening, accomplished by a convolution or filtering operation involving multiplication of transforms of data and filter kernel in the frequency domain, there is provided an inverse transformation resulting in a set of blocks of processed data samples. The blocks are overlapped followed by a savings of designated samples, and a discarding of excess samples from regions of overlap. The spatial representation of the kernel is modified by reduction of the number of components, for a linear-phase filter, and zero-padded to equal the number of samples of a data block, this being followed by forming the discrete odd cosine transform (DOCT) of the padded kernel matrix.

United States Patent No. 5,175,617 discloses a system and method for transmitting logmap video images through telephone line band-limited analog channels. The pixel organization in the logmap image is designed to match the sensor geometry of the human eye with a greater concentration of pixels at the center. The transmitter divides the frequency band into channels, and assigns one or two pixels to each channel, for example a 3KHz voice quality telephone

line is divided into 768 channels spaced about 3.9Hz apart. Each channel consists of two carrier waves in quadrature, so each channel can carry two pixels. Some channels are reserved for special calibration signals enabling the receiver to detect both the phase and magnitude of the received signal. If the sensor and pixels are connected directly to a bank of oscillators and the receiver can continuously receive each channel, then the receiver need not be synchronized with the transmitter. An FFT algorithm implements a fast discrete approximation to the continuous case in which the receiver synchronizes to the first frame and then acquires subsequent frames every frame period. The frame period is relatively low compared with the sampling period so the receiver is unlikely to lose frame synchrony once the first frame is detected. An experimental video telephone transmitted 4 frames per second, applied quadrature coding to 1440 pixel logmap images and obtained an effective data transfer rate in excess of 40,000 bits per second.

United States Patent No. 5,185,819 discloses a video compression system having odd and even fields of video signal that are independently compressed in sequences of intraframe and interframe compression modes. The odd and even fields of independently compressed data are interleaved for transmission such that the intraframe even field compressed data occurs midway between successive fields of intraframe odd field compressed data. The interleaved sequence provides receivers with twice the number of entry points into the signal for decoding without increasing the amount of data transmitted.

United States Patent No. 5,212,742 discloses an apparatus and method for processing video data for compression/decompression in real-time. The apparatus comprises a plurality of compute modules, in a preferred embodiment, for a total of four compute modules coupled in

parallel. Each of the compute modules has a processor, dual port memory, scratch-pad memory, and an arbitration mechanism. A first bus couples the compute modules and host processor. Lastly, the device comprises a shared memory
5 which is coupled to the host processor and to the compute modules with a second bus. The method handles assigning portions of the image for each of the processors to operate upon.

United States Patent No. 5,231,484 discloses a system
10 and method for implementing an encoder suitable for use with the proposed ISO/IEC MPEG standards. Included are three cooperating components or subsystems that operate to variously adaptively pre-process the incoming digital motion video sequences, allocate bits to the pictures in a sequence,
15 and adaptively quantize transform coefficients in different regions of a picture in a video sequence so as to provide optimal visual quality given the number of bits allocated to that picture.

United States Patent No. 5,267,334 discloses a method of
20 removing frame redundancy in a computer system for a sequence of moving images. The method comprises detecting a first scene change in the sequence of moving images and generating a first keyframe containing complete scene information for a first image. The first keyframe is known, in a preferred
25 embodiment, as a "forward-facing" keyframe or intraframe, and it is normally present in CCITT compressed video data. The process then comprises generating at least one intermediate compressed frame, the at least one intermediate compressed frame containing difference information from the first image
30 for at least one image following the first image in time in the sequence of moving images. This at least one frame being known as an interframe. Finally, detecting a second scene change in the sequence of moving images and generating a second keyframe containing complete scene information for an

image displayed at the time just prior to the second scene change, known as a "backward-facing" keyframe. The first keyframe and the at least one intermediate compressed frame are linked for forward play, and the second keyframe and the intermediate compressed frames are linked in reverse for reverse play. The intraframe may also be used for generation of complete scene information when the images are played in the forward direction. When this sequence is played in reverse, the backward-facing keyframe is used for the generation of complete scene information.

United States Patent No. 5,276,513 discloses a first circuit apparatus, comprising a given number of prior-art image-pyramid stages, together with a second circuit apparatus, comprising the same given number of novel motion-vector stages, perform cost-effective hierarchical motion analysis (HMA) in real-time, with minimum system processing delay and/or employing minimum system processing delay and/or employing minimum hardware structure. Specifically, the first and second circuit apparatus, in response to relatively high-resolution image data from an ongoing input series of successive given pixel-density image-data frames that occur at a relatively high frame rate (e.g., 30 frames per second), derives, after a certain processing-system delay, an ongoing output series of successive given pixel-density vector-data frames that occur at the same given frame rate. Each vector-data frame is indicative of image motion occurring between each pair of successive image frames.

United States Patent No. 5,283,646 discloses a method and apparatus for enabling a real-time video encoding system to accurately deliver the desired number of bits per frame, while coding the image only once, updates the quantization step size used to quantize coefficients which describe, for example, an image to be transmitted over a communications channel. The data is divided into sectors, each sector

including a plurality of blocks. The blocks are encoded, for example, using DCT coding, to generate a sequence of coefficients for each block. The coefficients can be quantized, and depending upon the quantization step, the number of bits required to describe the data will vary significantly. At the end of the transmission of each sector of data, the accumulated actual number of bits expended is compared with the accumulated desired number of bits expended, for a selected number of sectors associated with the particular group of data. The system then readjusts the quantization step size to target a final desired number of data bits for a plurality of sectors, for example describing an image. Various methods are described for updating the quantization step size and determining desired bit allocations.

The article, Chong, Yong M., A Data-Flow Architecture for Digital Image Processing, Wescon Technical Papers: No. 2 Oct./Nov. 1984, discloses a real-time signal processing system specifically designed for image processing. More particularly, a token based data-flow architecture is disclosed wherein the tokens are of a fixed one word width having a fixed width address field. The system contains a plurality of identical flow processors connected in a ring fashion. The tokens contain a data field, a control field and a tag. The tag field of the token is further broken down into a processor address field and an identifier field. The processor address field is used to direct the tokens to the correct data-flow processor, and the identifier field is used to label the data such that the data-flow processor knows what to do with the data. In this way, the identifier field acts as an instruction for the data-flow processor. The system directs each token to a specific data-flow processor using a module number (MN). If the MN matches the MN of the particular stage, then the appropriate operations are

performed upon the data. If unrecognized, the token is directed to an output data bus.

The article, Kimori, S. et al. An Elastic Pipeline Mechanism by Self-Timed Circuits, IEEE J. of Solid-State
5 Circuits, Vol. 23, No. 1, February 1988, discloses an elastic pipeline having self-timed circuits. The asynchronous pipeline comprises a plurality of pipeline stages. Each of the pipeline stages consists of a group of input data latches followed by a combinatorial logic circuit that carries out
10 logic operations specific to the pipeline stages. The data latches are simultaneously supplied with a triggering signal generated by a data-transfer control circuit associated with that stage. The data-transfer control circuits are interconnected to form a chain through which send and
15 acknowledge signal lines control a hand-shake mode of data transfer between the successive pipeline stages. Furthermore, a decoder is generally provided in each stage to select operations to be done on the operands in the present stage. It is also possible to locate the decoder in the
20 preceding stage in order to pre-decode complex decoding processing and to alleviate critical path problems in the logic circuit. The elastic nature of the pipeline eliminates any centralized control since all the interworkings between the submodules are determined by a completely localized
25 decision and, in addition, each submodule can autonomously perform data buffering and self-timed data-transfer control at the same time. Finally, to increase the elasticity of the pipeline, empty stages are interleaved between the occupied stages in order to ensure reliable data transfer between the
30 stages.

United States Patent No. 5,278,646 discloses an improved technique for decoding wherein the number of coefficients to be included in each sub-block is selectable, and a code indicating the number of coefficients within each layer is

097269340001

inserted in the bitstream at the beginning of each encoded video sequence. This technique allows the original runs of zero coefficients in the highest resolution layer to remain intact by forming a sub-block for each scale from a selected number of coefficients along a continuous scan. These sub-blocks may be decoded in a standard fashion, with an inverse discrete cosine transform applied to square sub-blocks obtained by the appropriate zero padding of and/or discarding of excess coefficients from each of the scales. This technique further improves decoding efficiency by allowing an implicit end of block signal to separate blocks, making it unnecessary to decode an explicit end of block signal in most cases.

United States Patent No. 4,903,018 discloses a process and data processing system for compressing and expanding structurally associated multiple data sequences. The process is particular to data sets in which an analysis is made of the structure in order to identify a characteristic common to a predetermined number of successive data elements of a data sequence. In place of data elements, a code is used which is again decoded during expansion. The common characteristic is obtained by analyzing data elements which have the same order number in a number of data sequences. During expansion, the data elements obtained by decoding the code are ordered in data series on the basis of the order number of these data series on the basis of the order number of these data elements. The data processing system for performing the processes includes a storage matrix (26) and an index storage (28) having line addresses of the storage matrix (26) in an assorted line sequence.

United States Patent No. 4,334,246 discloses a circuit and method for decompressing video subsequent to its prior compression for transmission or storage. The circuit assumes that the original video generated by a raster input scanner

99772693 "043004
FOUO 99272693

was operated on by a two line one shot predictor, coded using run length encoding into code words of four, eight or twelve bits and packed into sixteen bit data words. This described decompressor, then, unpacks the data by joining together the sixteen bit data words and then separately the individual code words, converts the code words into a number of all zero four bit nibbles and a terminating nibble containing one or more one bits which constitutes decoded data, inspects the actual video of the preceding scan line and the previous video bits of the present line to produce depredictor bits and compares the decoded data and depredictor bits to produce the final actual video.

United States Patent No. 5,060,242 discloses an image signal processing system DPCM encodes the signal, then Huffman and run length encodes the signal to produce variable length code words, which are then tightly packed without gaps for efficient transmission without loss of any data. The tightly packed apparatus has a barrel shifter with its shift modulus controlled by an accumulator receiving code word length information. An OR gate is connected to the shifter, while a register is connected to the gate. Apparatus for processing a tightly packed and decorrelated digital signal has a barrel shifter and accumulator for unpacking, a Huffman and run length decoder, and an inverse DCPM decoder.

United States Patent No. 5,168,375 discloses a method for processing a field of image data samples to provide for one or more of the functions of decimation, interpolation, and sharpening is accomplished by use of an array transform processor such as that employed in a JPEG compression system. Blocks of data samples are transformed by the discrete even cosine transform (DECT) in both the decimation and interpolation processes, after which the number of frequency terms is altered. In the case of decimation, the number of frequency terms is reduced, this being followed by inverse

FOR 862260

transformation to produce a reduced-size matrix of sample points representing the original block of data. In the case of interpolation, additional frequency components of zero value are inserted into the array of frequency components after which inverse transformation produces an enlarged data sampling set without an increase in spectral bandwidth. In the case of sharpening, accomplished by a convolution or filtering operation involving multiplication of transforms of data and filter kernel in the frequency domain, there is provided an inverse transformation resulting in a set of blocks of processed data samples. The blocks are overlapped followed by a savings of designated samples, and a discarding of excess samples from regions of overlap. The spatial representation of the kernel is modified by reduction of the number of components, for a linear-phase filter, and zero-padded to equal the number of samples of a data block, this being followed by forming the discrete odd cosine transform (DOCT) of the padded kernel matrix.

United States Patent No. 5,231,486 discloses a high definition video system processes a bitstream including high and low priority variable length coded Data words. The coded Data is separated into packed High Priority Data and packed Low Priority Data by means of respective data packing units. The coded Data is continuously applied to both packing units. High Priority and Low Priority Length words indicating the bit lengths of high priority and low priority components of the coded Data are applied to the high and low priority data packers, respectively. The Low Priority Length word is zeroed when high Priority Data is to be packed for transport via a first output path, and the High Priority Length word is zeroed when Low Priority Data is to be packed for transport via a second output path.

United States Patent No. 5,287,178 discloses a video signal encoding system includes a signal processor for

segmenting encoded video data into transport blocks having a header section and a packed data section. The system also includes reset control apparatus for releasing resets of system components, after a global system reset, in a prescribed non-simultaneous phased sequence to enable signal processing to commence in the prescribed sequence. The phased reset release sequence begins when valid data is sensed as transmitting the data lines.

United States Patent No. 5,124,790 to Nakayama discloses a reverse quantizer to be used with image memory. The inverse quantizer is used in the standard way to decode differential predictive coding method (DPCM) encoded data.]

United States Patent No. 5,136,371 to Savatier et al. is directed to a de-quantizer having an adjustable quantizational level which is variable and determined by the fullness of the buffer. The applicants state that the novel aspect of their invention is the maximum available data rate that is achieved. Buffer overflow and underflow is avoided by adapting the quantization step size the quantizer 152 and the de-quantizer 156 by means of a quantizational level which is recalculated after each block has been encoded. The quantization level is calculated as a function of the amount of already encoded data for the frame, compared with the total buffer size. In this manner, the quantization level can advantageously be recalculated by the decoder and does not have to be transmitted.

United States Patent No. 5,142,380 to Sakagami et al. discloses an image compression apparatus suitable for use with still images such as those formed by electronic still cameras using solid state image sensors. The quantizer employed is connected to a memory means from which threshold values of a quantization matrix for the laminate signal, Y, and rom 15 stores threshold values of a quantization matrix for the crominant signals I and Q.

United States Patent No. 5,193,002 to Guichard et al. disclosed an apparatus for coding/decoding image signals in real time in conjunction with the CCITT standard H.261. A digital signal processor carries out direct quantization and reverse quantization.

United States Patent No. 5,241,383 to Chen et al. describes an apparatus with a pseudo-constant bit rate video coding achieved by an adjustable quantization parameter. The quantization parameter utilized by the quantizer 32 is periodically adjusted to increase or decrease the amount of code bits generated by the coding circuit. The change in quantization parameters for coding the next group of pictures is determined by a deviation measure between the actual number of code bits generated by the coding circuits for the previous group of pictures in an estimate number of code bits for the previous group of pictures. The number of code bits generated by the coding circuit is controlled by controlling the quantizer step sizes. In general smaller quantizer step sizes result in more code bits in larger quantizer step sizes result in fewer code bits.

United States Patent No. 5,113,255 to Nagata et al; 5,126,842 to Andrews et al; 5,253,058 to Gharavi; 5,260,782 to Hui; and 5,212,742 to Normile et al are included for background and as a general description of the art.

Accordingly, those concerned with the design, development and use of video compression/decompression systems and related subsystems have long recognized a need for improved methods and apparatus providing enhanced flexibility, efficiency and performance. The present invention clearly fulfills all these needs.

SUMMARY OF THE INVENTION

Briefly, and in general terms, the present invention provides an input, an output and a plurality of processing stages between the input and the output, the plurality of processing stages being interconnected by a two-wire interface for conveyance of tokens along a pipeline, and control and/or DATA tokens in the form of universal adaptation units for interfacing with all of the stages in the pipeline and interacting with selected stages in the pipeline for control, data and/or combined control-data functions among the processing stages, whereby the processing stages in the pipeline are afforded enhanced flexibility in configuration and processing.

Each of the processing stages in the pipeline may include both primary and secondary storage, and the stages in the pipeline are reconfigurable in response to recognition of selected tokens. The tokens in the pipeline are dynamically adaptive and may be position dependent upon the processing stages for performance of functions or position independent of the processing stages for performance of functions.

In a pipeline machine, in accordance with the invention, the tokens may be altered by interfacing with the stages, and the tokens may interact with all of the processing stages in the pipeline or only with some but less than all of said processing stages. The tokens in the pipeline may interact with adjacent processing stages or with non-adjacent processing stages, and the tokens may reconfigure the processing stages. Such tokens may be position dependent for some functions and position independent for other functions in the pipeline.

The tokens, in combination with the reconfigurable processing stages, provide a basic building block for the pipeline system. The interaction of the tokens with a processing stage in the pipeline may be conditioned by the previous processing history of that processing stage. The

tokens may have address fields which characterize the tokens, and the interactions with a processing stage may be determined by such address fields.

5 In an improved pipeline machine, in accordance with the invention, the tokens may include an extension bit for each token, the extension bit indicating the presence of additional words in that token and identifying the last word in that token. The address fields may be of variable length
10 and may also be Huffman coded.

15 In the improved pipeline machine, the tokens may be generated by a processing stage. Such pipeline tokens may include data for transfer to the processing stages or the tokens may be devoid of data. Some of the tokens may be identified as DATA tokens and provide data to the processing
20 stages in the pipeline, while other tokens are identified as control tokens and only condition the processing stages in the pipeline, such conditioning including reconfiguring of the processing stages. Still other tokens may provide both data and conditioning to the processing stages in the
25 pipeline. Some of said tokens may identify coding standards to the processing stages in the pipeline, whereas other tokens may operate independent of any coding standard among the processing stages. The tokens may be capable of successive alteration by the processing stages in the pipeline.

30 In accordance with the invention, the interactive flexibility of the tokens in cooperation with the processing stages facilitates greater functional diversity of the processing stages for resident structure in the pipeline, and the flexibility of the tokens facilitates system expansion
35 and/or alteration. The tokens may be capable of facilitating a plurality of functions within any processing stage in the pipeline. Such pipeline tokens may be either hardware based or software based. Hence, the tokens facilitate more

efficient uses of system bandwidth in the pipeline. The tokens may provide data and control simultaneously to the processing stages in the pipeline.

5 The invention may include a pipeline processing machine for handling plurality of separately encoded bit streams arranged as a single serial bit stream of digital bits and having separately encoded pairs of control codes and corresponding data carried in the serial bit stream and
10 employing a plurality of stages interconnected by a two-wire interface, further characterized by a start code detector responsive to the single serial bit stream for generating control tokens and DATA tokens for application to the two-wire interface, a token decode circuit positioned in certain
15 of the stages for recognizing certain of the tokens as control tokens pertinent to that stage and for passing unrecognized control tokens along the pipeline, and a reconfigurable decode and parser processing means responsive to a recognized control token for reconfiguring a particular
20 stage to handle an identified DATA token.

 The pipeline machine may also include first and second registers, the first register being positioned as an input of the decode and parser means, with the second register positioned as an output of the decode and parser means. One
25 of the processing stages may be a spatial decoder, a second of the stages being a token generator for generating control tokens and DATA tokens for passage along the two-wire interface. A token decode means is positioned in the spatial decoder for recognizing certain of the tokens as control
30 tokens pertinent to the spatial decoder and for configuring the spatial decoder for spatially decoding DATA tokens following a control token into a first decoded format.

 A further stage may be a temporal decoder positioned downstream in the pipeline from the spatial decoder, with a
35 second token decode means positioned in the temporal decoder

for recognizing certain of the tokens as control tokens
pertinent to the temporal decoder and for configuring the
temporal decoder for temporally decoding the DATA tokens
5 following the control token into a first decoded format. The
temporal decoder may utilize a reconfigurable prediction
filter which is reconfigurable by a prediction token.

Data may be moved along the two-wire interface within
the temporal decoder in 8x8 pel data blocks, and address
10 means may be provided for storing and retrieving such data
blocks along block boundaries. The address means may store
and retrieve blocks of data across block boundaries. The
address means reorders said blocks as picture data for
display. The data blocks stored and retrieved may be greater
15 and/or smaller than 8x8 pel data blocks. Circuit means may
also be provided for either displaying the output of the
temporal decoder or writing the output back into a picture
memory location. The decoded format may be either a still
picture format or a moving picture format.

20 The processing stage may also include, in accordance
with the invention, a token decoder for decoding the address
of a token and an action identifier responsive to the token
decoder to implement configuration of the processing stage.
The processing stages reside in a pipeline processing machine
25 having a plurality of the processing stages interconnected by
a two-wire interface bus, with control tokens and DATA tokens
passing over the two-wire interface. A token decode circuit
is positioned in certain of the processing stages for
recognizing certain of the tokens as control tokens pertinent
30 to that stage and for passing unrecognized control tokens
along the pipeline. A first input latch circuit may be
positioned on the two-wire interface preceding the processing
stage and a second output latch circuit may be positioned on
the two-wire interface succeeding the processing stage. The
35 token decode circuit is connected to the two-wire interface

through the first input latch. Predetermined processing stages may include a decoding circuit connected to the output of a predetermined data storage device, whereby each processing stage assumes the active state only when the stage contains a predetermined stage activation signal pattern and remains in the activation mode until the stage contains a predetermined stage deactivation pattern.

In accordance with the invention, one of the stages is a Start Code Detector for receiving the input and being adapted to generate and/or convert the tokens. The Start Code Detector is responsive to data to create tokens, searches for and detects start codes and produces tokens in response thereto, and is capable of detecting overlapping start codes, whereby the first start code is ignored and the second start code is used to create start code tokens.

The Start Code Detector stage is adapted to search an input data stream in a search mode for a selected start code. The detector searches for breaks in the data stream, and the search may be made of data from an external data source. The Start Code Detector stage may produce a START CODE token, a PICTURE_START token, a SLICE_START token, a PICTURE_END token, a SEQUENCE_START token, a SEQUENCE_END token, and/or a GROUP_START token. The Start Code Detector stage may also perform a padding function by adding bits to the last word of a token.

The Start Code Detector may provide, in a machine for handling a plurality of separately encoded bit streams arranged as a serial bit stream of digital bits and having separately encoded pairs of start codes and data carried in the serial bit stream, a Start Code Detector subsystem having first, second and third registers connected in serial fashion, each of the registers storing a different number of bits from the bit stream, the first register storing a value, the second register and a first decode means identifying a

start code associated with the value contained in said first register. Circuit means shift the latter value to a predetermined end of the third register, and a second decode means is arranged for accepting data from the third register in parallel.

A memory may also be provided which is responsive to the second decode means for providing one or more control tokens stored in the memory as a result of the decoding of the value associated with the start code. A plurality of tag shift registers may be provided for handling tags indicating the validity of data from the registers. The system may also include means for accessing the input data stream from a microprocessor interface, and means for formatting and organizing the data stream.

In accordance with the invention, the Start Code Detector may identify start codes of varying widths associated with differently encoded bit streams. The detector may generate a plurality of DATA Tokens from the input data stream. Further in accordance with the invention, the system may be a pipeline system and the Start Code Detector may be positioned as the first processing stage in the pipeline.

The present invention also provides, in a digital picture information processing system, means for selectively configuring the system to process data in accordance with a plurality of different picture compression/decompression standards. The picture standards may include JPEG, MPEG, and/or H.261, or any other standards and any combination of such picture standards, without departing in any way from the spirit and scope of the invention. In accordance with the invention, the system may include a spatial decoder for video data and having a Huffman decoder, an index to data and an arithmetic logic unit with a microcode ROM having separate stored programs for each of a plurality of different picture

compression/decompression standards, such programs being selectable by an interfacing adaptation unit in the form of a token, so that processing for a plurality of picture standards is facilitated. A multi-standard system in accordance with the invention, may utilize tokens for its operation regardless of the selected picture standard, and the tokens may be utilized as a generic communication protocol in the system for all of the various picture standards. The system may be further characterized by a multi-standard token for mapping differently encoded data streams arranged on a single serial stream of data onto a single decoder using a mixture of standard dependent and standard independent hardware and control tokens. The system may also include an address generation means for arranging macroblocks of data associated with different picture standards into a common addressing scheme.

The present invention also provides, in a system having a plurality of processing stages, a universal adaptation unit in the form of an interactive interfacing token for control and/or data functions among the processing stages, the token being a PICTURE_START code token for indicating that the start of a picture will follow in the subsequent DATA token.

The token may also be a PICTURE_END token for indicating the end of an individual picture.

The token may also be a FLUSH token for clearing buffers and resetting the system as it proceeds down the system from the input to the output. In accordance with the invention, the FLUSH token may variably reset the stages as the token proceeds down the pipeline.

The token may also be a CODING_STANDARD token for conditioning the system for processing in a selected one of a plurality of picture compression/decompression standards.

The CODING_STANDARD token may designate the picture standard as JPEG, and/or any other appropriate picture

standard. At least some of the processing stages reconfigure in response to the CODING_STANDARD token.

One of the processing stages in the system may be a Huffman decoder and parser and, upon receipt of a CODING_STANDARD control token, the parser is reset to an address location corresponding to the location of a program for handling the picture standard identified by the CODING_STANDARD control token. A reset address may also be selected by the CODING_STANDARD control token corresponding to a memory location used for testing the Huffman decoder and parser.

The Huffman decoder may include a decoding stage and an Index to Data stage, and the parser stage may send an instruction to the Index to Data Unit to select tables needed for a particular identified coding standard, the parser stage indicating whether the arriving data is inverted or not.

The aforescribed tokens may take the form of an interactive metamorphic interfacing token.

The present invention also provides a system for decoding video data, having a Huffman decoder, an index to data (ITOD) stage, an arithmetic logic unit (ALU), and a data buffering means immediately following the system, whereby time spread for video pictures of varying data size can be controlled.

The system may include a spatial decoder having a two-wire interface interconnecting processing stages, the interface enabling serial processing for data and parallel processing for control.

As previously indicated, the system may further include a ROM having separate stored programs for each of a plurality of picture standards, the programs being selectable by a token to facilitate processing for a plurality of different picture standards.

The spatial decoder system also includes a token formatter for formatting tokens, so that DATA tokens are created.

5 The system may also include a decoding stage and a parser stage for sending an instruction to the Index to Data Unit to select tables needed for a particular identified coding standard, the parser stage indicating whether the arriving data is inverted or not. The tables may be arranged
10 within a memory for enabling multiple use of the tables where appropriate.

 The present invention also provides a pipeline system having an input data stream, and a processing stage for receiving the input data stream, the stage including means
15 for recognizing specified bit stream patterns, whereby said stage facilitates random access and error recovery. In accordance with the invention, the processing stage may be a start code detector and the bit stream patterns may include start codes. Hence, the invention provides a search-mode
20 means for searching differently encoded data streams arranged as a single serial stream of data for allowing random access and enhanced error recovery.

 The present invention also provides a pipeline machine having means for performing a stop-after-picture operation
25 for achieving a clear end to picture data decoding, for indicating the end of a picture, and for clearing the pipeline, wherein such means generates a combination of a PICTURE_END token and a FLUSH token.

 The present invention also provides, in a pipeline
30 machine, a fixed size, fixed width buffer and means for padding the buffer to pass an arbitrary number of bits through the buffer. The padding means may be a start code detector.

 Padding may be performed only on the last word of a
35 token and padding insures uniformity of word size. In

accordance with the invention, a reconfigurable processing stage may be provided as a spatial decoder and the padding means adds to picture data being handled by the spatial decoder sufficient additional bits such that each decompressed picture at the output of the spatial decoder is of the same length in bits.

The present invention also provides, in a system having a data stream including run length code, an inverse modeller means active upon the data stream from a token for expending out the run level code to a run of zero data followed by a level, whereby each token is expressed with a specified number of values. The token may be a DATA token.

The inverse modeller means blocks tokens which lack the specified number of values, and the specified number of values may be 64 coefficients in a presently preferred embodiment of the invention.

The practice of the invention may include an expanding circuit for accepting a DATA token having run length codes and decoding the run length codes. A padder circuit in communication with the expanding circuit checks that the DATA token has a predetermined length so that if the DATA token has less than the predetermined length, the padder circuit adds units of data to the DATA token until the predetermined length is achieved. A bypass circuit is also provided for bypassing any token other than a DATA token around the expanding circuit and the padding circuit.

In accordance with the invention, a method is provided for data to efficiently fill a buffer, including providing first type tokens having a first predetermined width, and at least one of the following formats:

Format A - ExxxxxxLLLLLLLLLLLLL

Format B - ERRRRRRLLLLLLLLLLLLLL

Format C - E000000LLLLLLLLLLLLL

where E=extension bit; F=specifics format; R=run bit;

L=length bit or non-data token; x="don't care" bit, splitting format A tokens into a format 0a token having a form of ELLLLLLLLLLLL, splitting format B tokens into a format 1 token
 5 having the form of FRRRRRRR00000 and a format 0a data token, splitting format C tokens into a format 0 token having the form of FLLLLLLLLLLLL, and packing format 0, format 0a and format 1 tokens into a buffer, having a second predetermined width.

10 The invention also provides an apparatus for providing a time delay to a group of compressed pictures, the pictures corresponding to a video compression/ decompression standard, wherein words of data containing compressed pictures are counted by a counter circuit and a microprocessor, in
 15 communication with the counter circuit and adapted to receive start-up information consistent with the standard of video decompression, communicates the start-up information to the counter circuit.

An inverse modeller circuit, for accepting the words of
 20 data and capable of delaying the words of data, is in communication with a control circuit intermediate the counter circuit and the inverse modeller circuit, the control circuit also communicating with the counter circuit which compares the start-up information with the counted words of data and
 25 signals the control circuit. The control circuit queues the signals in correspondence to the words of data that have met the start-up criterion and controls the inverse modeller delay feature. .

The present invention also provides in a pipeline system
 30 having an inverse modeller stage and an inverse discrete cosine transform stage, the improvement characterized by a processing stage, positioned between the inverse modeller stage and the inverse discrete cosine transform stage, responsive to a token table for processing data.

In accordance with the invention, the token may be a QUANT_TABLE token for causing the processing stage to generate a quantization table.

5 The present invention also provides a Huffman decoder for decoding data words encoded according to the Huffman coding provisions of either H.261, JPEG or MPEG standards, the data words including an identifier that identifies the Huffman code standard under which the data words were coded,
10 and comprising means for receiving the Huffman coded data words, means for reading the identifier to determine which standard governed the Huffman coding of the received data words, means for converting the data words to JPEG Huffman coded data words, if necessary, in response to reading the
15 identifier that identifies the Huffman coded data words as H.261 or MPEG Huffman coded, means operably connected to the Huffman coded data words receiving means for generating an index number associated with each JPEG Huffman coded data word received from the Huffman coded data words receiving
20 means, and means for operating a lookup table containing a Huffman code table having the format used under the JPEG standard to transmit JPEG Huffman table information, including an input for receiving an index number from the index number generating means, and including an output that
25 is a decoded data word corresponding to the index number.

 The invention further relates, in varying degrees of scope, to a method for decoding data words encoded according to the Huffman coding provisions of either H.261, JPEG or MPEG standards, the data words including an identifier that
30 identifies the Huffman code standard under which the data words were coded, such steps comprising receiving the Huffman coded data words, including reading the identifier to determine which standard governed the Huffman coding of the received data words, if necessary, in response to reading the
35 identifier that identifies the Huffman coded data words as

5

10

DESCRIPTION OF THE DRAWINGS

- Figure. 1 illustrates six cycles of a six-stage pipeline for different combinations of two internal control signals;
- 5 Figures. 2a and 2b illustrate a pipeline in which each stage includes auxiliary data storage. They also show the manner in which pipeline stages can "compress" and "expand" in response to delays in the pipeline;
- Figures. 3a(1), 3a(2), 3b(1) and 3b(2) illustrate the control
- 10 of data transfer between stages of a preferred embodiment of a pipeline using a two-wire interface and a multi-phase clock;
- Figure. 4 is a block diagram that illustrates a basic embodiment of a pipeline stage that incorporates a two-wire
- 15 transfer control and also shows two consecutive pipeline processing stages with the two-wire transfer control;
- Figures. 5a and 5b taken together depict one example of a timing diagram that shows the relationship between timing signals, input and output data, and internal control signals
- 20 used in the pipeline stage as shown in Figure. 4;
- Figure. 6 is a block diagram of one example of a pipeline stage that holds its state under the control of an extension bit;
- Figure. 7 is a block diagram of a pipeline stage that decodes
- 25 stage activation data words;
- Figures. 8a and 8b taken together form a block diagram showing the use of the two-wire transfer control in an exemplifying "data duplication" pipeline stage;
- Figures. 9a and 9b taken together depict one example of a
- 30 timing diagram that shows the two-phase clock, the two-wire transfer control signals and the other internal data and control signals used in the exemplifying embodiment shown in Figures. 8a and 8b.
- Figure 10 is a block diagram of a reconfigurable processing
- 35 stage;

offset by (1,1);

Figure 27 is a block diagram illustrating the Huffman decoder and parser state machine of the Spatial Decoder.

- 5 Figure 28 is a block diagram illustrating the prediction filter.

Figure 27

FIGURES

| | | |
|----|---------------------------------|---|
| | Figure 29 | shows a typical decoder system; |
| | Figure 30 | shows a JPEG still picture decoder; |
| 5 | Figure 31 | shows a JPEG video decoder; |
| | Figure 32 | shows a multi-standard video decoder; |
| | Figure 33 | shows the start and the end of a token; |
| | Figure 34 | shows a token address and data fields; |
| | Figure 35 | shows a token on an interface wider than |
| 10 | 8 bits; | |
| | Figure 36 | shows a macroblock structure; |
| | Figure 37 | shows a two-wire interface protocol; |
| | Figure 38 | shows the location of external two-wire |
| | interfaces; | |
| 15 | Figure 39 | shows clock propagation; |
| | Figure 40 | shows two-wire interface timing; |
| | Figure 41 | shows examples of access structure; |
| | Figure 42 | shows a read transfer cycle; |
| | Figure 43 | shows an access start timing; |
| 20 | Figure 44 | shows an example access with two write |
| | transfers; | |
| | Figure 45 | shows a read transfer cycle; |
| | Figure 46 | shows a write transfer cycle; |
| | Figure 47 | shows a refresh cycle; |
| 25 | Figure 48 | shows a 32 bit data bus and a 256 kbit |
| | deep DRAMs (9 bit row address); | |
| | Figure 49 | shows timing parameters for any strobe |
| | signal; | |
| | Figure 50 | shows timing parameters between any two |
| 30 | strobe signals; | |
| | Figure 51 | shows timing parameters between a bus and |
| | a strobe; | |
| | Figure 52 | shows timing parameters between a bus and |
| | a strobe; | |
| 35 | Figure 53 | shows an MPI read timing; |
| | Figure 54 | shows an MPI write timing; |
| | Figure 55 | shows organization of large integers in |
| | the memory map; | |
| | Figure 56 | shows a typical decoder clock regime; |
| 40 | Figure 57 | shows input clock requirements; |
| | Figure 58 | shows the Spatial Decoder; |
| | Figure 59 | shows the inputs and outputs of the input |
| | circuit; | |
| | Figure 60 | shows the coded port protocol; |
| 45 | Figure 61 | shows the start code detector; |
| | Figure 62 | shows start codes detected and converted |
| | to Tokens; | |
| | Figure 63 | shows the start codes detector passing |
| | Tokens; | |
| 50 | Figure 64 | shows overlapping MPEG start codes (byte |
| | aligned); | |
| | Figure 65 | shows overlapping MPEG start codes (not |

| | | |
|----|-----------------------|---|
| | byte aligned); | |
| | Figure 66 | shows jumping between two video |
| | sequences; | |
| 5 | Figure 67 | shows a sequence of extra Token |
| | insertion; | |
| | Figure 68 | shows decoder start-up control; |
| | Figure 69 | shows enabled streams queued before the |
| | output; | |
| 10 | Figure 70 | shows a spatial decoder buffer; |
| | Figure 71 | shows a buffer pointer; |
| | Figure 72 | shows a video demux; |
| | Figure 73 | shows a construction of a picture; |
| | Figure 74 | shows a construction of a 4:2:2 |
| 15 | macroblock; | |
| | Figure 75 | shows a calculating macroblock dimension |
| | from pel ones; | |
| | Figure 76 | shows spatial decoding; |
| | Figure 77 | shows an overview of H.261 inverse |
| 20 | quantization; | |
| | Figure 78 | shows an overview of JPEG inverse |
| | quantization; | |
| | Figure 79 | shows an overview of MPEG inverse |
| | quantization; | |
| 25 | Figure 80 | shows a quantization table memory map; |
| | Figure 81 | shows an overview of JPEG baseline |
| | sequential structure; | |
| | Figure 82 | shows a tokenised JPEG picture; |
| | Figure 83 | shows a temporal decoder; |
| 30 | Figure 84 | shows a picture buffer specification; |
| | Figure 85 | shows an MPEG picture sequence (m=3); |
| | Figure 86 | shows how "I" pictures are stored and |
| | output; | |
| | Figure 87 | shows how "P" pictures are formed, stored |
| 35 | and output; | |
| | Figure 88 | shows how "B" pictures are formed and |
| | output; | |
| | Figure 89 | shows P picture formation; |
| | Figure 90 | shows H.261 prediction formation; |
| 40 | Figure 91 | shows an H.261 "sequence"; |
| | Figure 92 | shows a hierarchy of H.261 syntax; |
| | Figure 93 | shows an H.261 picture layer; |
| | Figure 94 | shows an H.261 arrangement of groups of |
| | blocks; | |
| 45 | Figure 95 | shows an H.261 "slice" layer; |
| | Figure 96 | shows an H.261 arrangement of |
| | macroblocks; | |
| | Figure 97 | shows an H.261 sequence of blocks; |
| | Figure 98 | shows an H.261 macroblock layer; |
| 50 | Figure 99 | shows an H.261 arrangement of pels in |
| | blocks; | |
| | Figure 100 | shows a hierarchy of MPEG syntax; |
| | Figure 101 | shows an MPEG sequence layer; |

| | | |
|----|------------|---|
| | Figure 102 | shows an MPEG group of pictures layer; |
| | Figure 103 | shows an MPEG picture layer; |
| | Figure 104 | shows an MPEG "slice" layer; |
| 5 | Figure 105 | shows an MPEG sequence of blocks; |
| | Figure 106 | shows an MPEG macroblock layer; |
| | Figure 107 | shows an "open GOP"; |
| | Figure 108 | shows examples of access structure; |
| | Figure 109 | shows access start timing; |
| 10 | Figure 110 | shows a fast page read cycle; |
| | Figure 111 | shows a fast page write cycle; |
| | Figure 112 | shows a refresh cycle; |
| | Figure 113 | shows extracting row and column address from a chip address; |
| 15 | Figure 114 | shows timing parameters for any strobe signal; |
| | Figure 115 | shows timing parameters between any two strobe signals; |
| | Figure 116 | shows timing parameters between a bus and a strobe; |
| 20 | Figure 117 | shows timing parameters between a bus and a strobe; |
| | Figure 118 | shows a Huffman decoder and parser; |
| | Figure 119 | shows an H.261 and an MPEG AC Coefficient Decoding Flow Chart; |
| 25 | Figure 120 | shows a block diagram for JPEG (AC and DC) coefficient decoding; |
| | Figure 121 | shows a flow diagram for JPEG (AC and DC) coefficient decoding; |
| 30 | Figure 122 | shows an interface to the Huffman Token Formatter; |
| | Figure 123 | shows a token formatter block diagram; |
| | Figure 124 | shows an H.261 and an MPEG AC Coefficient Decoding; |
| 35 | Figure 125 | shows the interface to the Huffman ALU; |
| | Figure 126 | shows the basic structure of the Huffman ALU; |
| | Figure 127 | shows the buffer manager; |
| | Figure 128 | shows an imodel and hspk block diagram; |
| 40 | Figure 129 | shows an imex state diagram; |
| | Figure 130 | illustrates the buffer start-up; |
| | Figure 131 | shows a DRAM interface; |
| | Figure 132 | shows a write swing buffer; |
| | Figure 133 | shows an arithmetic block; |
| 45 | Figure 134 | shows an iq block diagram; |
| | Figure 135 | shows an iqca state machine; |
| | Figure 136 | shows an IDCT 1-D Transform Algorithm; |
| | Figure 137 | shows an IDCT 1-D Transform Architecture; |
| | Figure 138 | shows a token stream block diagram; |
| 50 | Figure 139 | shows a standard block structure; |
| | Figure 140 | is a block diagram showing; microprocessor test access; |
| | Figure 141 | shows 1-D Transform Micro-Architecture; |

| | | |
|----|---------------------------------|--|
| | Figure 142 | shows a temporal decoder block diagram; |
| | Figure 143 | shows the structure of a Two-wire |
| | interface stage; | |
| 5 | Figure 144 | shows the address generator block |
| | diagram; | |
| | Figure 145 | shows the block and pixel offsets; |
| | Figure 146 | shows multiple prediction filters; |
| | Figure 147 | shows a single prediction filter; |
| 10 | Figure 148 | shows the 1-D prediction filter; |
| | Figure 149 | shows a block of pixels; |
| | Figure 150 | shows the structure of the read rudder; |
| | Figure 151 | shows the block and pixel offsets; |
| | Figure 152 | shows a prediction example; |
| 15 | Figure 153 | shows the read cycle; |
| | Figure 154 | shows the write cycle; |
| | Figure 155 | shows the top-level registers block |
| | diagram with timing references; | |
| | Figure 156 | shows the control for incrementing |
| 20 | presentation numbers; | |
| | Figure 157 | shows the buffer manager state machine |
| | (complete); | |
| | Figure 158 | shows the state machine main loop; |
| | Figure 159 | shows the buffer 0 containing an SIF (22 |
| 25 | by 18 macroblocks) picture; | |
| | Figure 160 | shows the SIF component 0 with a display |
| | window; | |
| | Figure 161 | shows an example picture format showing |
| | storage block address; | |
| 30 | Figure 162 | shows a buffer 0 containing a SIF (22 by |
| | 18 macroblocks) picture; | |
| | Figure 163 | shows an example address calculation; |
| | Figure 164 | shows a write address generation state |
| | machine; | |
| 35 | Figure 165 | shows a slice of the datapath; |
| | Figure 166 | shows a two cycle operation of the |
| | datapath; | |
| | Figure 167 | shows mode 1 filtering; |
| | Figure 168 | shows a horizontal up-sampler datapath; |
| 40 | and | |
| | Figure 169 | shows the structure of the color-space |
| | converter. | |

In the ensuing description of the practice of the invention, the following terms are frequently used and are generally defined by the following glossary:

5

GLOSSARY

BLOCK: An 8-row by 8-column matrix of pels, or 64 DCT coefficients (source, quantized or dequantized).

CHROMINANCE (COMPONENT): A matrix, block or single pel representing one of the two color difference signals related to the primary colors in the manner defined in the bit stream. The symbols used for the color difference signals are Cr and Cb.

CODED REPRESENTATION: A data element as represented in its encoded form.

15 **CODED VIDEO BIT STREAM:** A coded representation of a series of one or more pictures as defined in this specification.

CODED ORDER: The order in which the pictures are transmitted and decoded. This order is not necessarily the same as the display order.

20 **COMPONENT:** A matrix, block or single pel from one of the three matrices (luminance and two chrominance) that make up a picture.

COMPRESSION: Reduction in the number of bits used to represent an item of data.

25 **DECODER:** An embodiment of a decoding process.

DECODING (PROCESS): The process defined in this specification that reads an input coded bitstream and produces decoded pictures or audio samples.

DISPLAY ORDER: The order in which the decoded pictures are displayed. Typically, this is the same order in which they were presented at the input of the encoder.

30 **ENCODING (PROCESS):** A process, not specified in this specification, that reads a stream of input pictures or audio samples and produces a valid coded bitstream as defined in this specification.

35

INTRA CODING: Coding of a macroblock or picture that uses information only from that macroblock or picture.

LUMINANCE (COMPONENT): A matrix, block or single pel representing a monochrome representation of the signal and related to the primary colors in the manner defined in the bit stream. The symbol used for luminance is Y.

MACROBLOCK: The four 8 by 8 blocks of luminance data and the two (for 4:2:0 chroma format) four (for 4:2:2 chroma format) or eight (for 4:4:4 chroma format) corresponding 8 by 8 blocks of chrominance data coming from a 16 by 16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the pel data and sometimes to the coded representation of the pel values and other data elements defined in the macroblock header of the syntax defined in this part of this specification. To one of ordinary skill in the art, the usage is clear from the context.

MOTION COMPENSATION: The use of motion vectors to improve the efficiency of the prediction of pel values. The prediction uses motion vectors to provide offsets into the past and/or future reference pictures containing previously decoded pel values that are used to form the prediction error signal.

MOTION VECTOR: A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture to the coordinates in a reference picture.

NON-INTRA CODING: Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.

PEL: Picture element.

PICTURE: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. For progressive video, a picture is

identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.

- 5 **PREDICTION:** The use of a predictor to provide an estimate of the pel value or data element currently being decoded.
- RECONFIGURABLE PROCESS STAGE (RPS):** A stage, which in response to a recognized token, reconfigures itself to perform various operations.
- 10 **SLICE:** A series of macroblocks.
- TOKEN:** A universal adaptation unit in the form of an interactive interfacing messenger package for control and/or data functions.
- START CODES [SYSTEM AND VIDEO]:** 32-bit codes embedded in a coded bitstream that are unique. They are used for several purposes including identifying some of the structures in the coding syntax.
- 15 **VARIABLE LENGTH CODING; VLC:** A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.
- 20 **VIDEO SEQUENCE:** A series of one or more pictures.
- Detailed Descriptions

DESCRIPTION OF THE PREFERRED EMBODIMENT(S)

As an introduction to the most general features used in a pipeline system which is utilized in the preferred
5 embodiments of the invention, Fig. 1 is a greatly simplified illustration of six cycles of a six-stage pipeline. (As is explained in greater detail below, the preferred embodiment of the pipeline includes several advantageous features not shown in Fig 1.).

10 Referring now to the drawings, wherein like reference numerals denote like or corresponding elements throughout the various figures of the drawings, and more particularly to Fig. 1, there is shown a block diagram of six cycles in practice of the present invention. Each row of boxes
15 illustrates a cycle and each of the different stages are labelled A-F, respectively. Each shaded box indicates that the corresponding stage holds valid data, i.e., data that is to be processed in one of the pipeline stages. After processing (which may involve nothing more than a simple
20 transfer without manipulation of the data) valid data is transferred out of the pipeline as valid output data.

Note that an actual pipeline application may include more or fewer than six pipeline stages. As will be appreciated, the present invention may be used with any number of pipeline
25 stages. Furthermore, data may be processed in more than one stage and the processing time for different stages can differ.

In addition to clock and data signals (described below), the pipeline includes two transfer control signals -- a
30 "VALID" signal and an "ACCEPT" signal. These signals are used to control the transfer of data within the pipeline. The VALID signal, which is illustrated as the upper of the two lines connecting neighboring stages, is passed in a forward or downstream direction from each pipeline stage to
35 the nearest neighboring device. This device may be another

pipeline stage or some other system. For example, the last pipeline stage may pass its data on to subsequent processing circuitry. The ACCEPT signal, which is illustrated as the lower of the two lines connecting neighboring stages, passes
5 in the other direction upstream to a preceding device.

A data pipeline system of the type used in the practice of the present invention has, in preferred embodiments, one or more of the following characteristics:

1. The pipeline is "elastic" such that a delay at a
10 particular pipeline stage causes the minimum disturbance possible to other pipeline stages. Succeeding pipeline stages are allowed to continue processing and, therefore, this means that gaps open up in the stream of data following the delayed stage. Similarly, preceding
15 pipeline stages may also continue where possible. In this case, any gaps in the data stream may, wherever possible, be removed from the stream of data.
2. Control signals that arbitrate the pipeline are organized so that they only propagate to the nearest
20 neighboring pipeline stages. In the case of signals flowing in the same direction as the data flow, this is the immediately succeeding stage. In the case of signals flowing in the opposite direction to the data flow, this is the immediately preceding stage.
- 25 3. The data in the pipeline is encoded such that many different types of data are processed in the pipeline. This encoding accommodates data packets of variable size and the size of the packet need not be known in advance.
4. The overhead associated with describing the type of
30 data is as small as possible.
5. It is possible for each pipeline stage to recognize only the minimum number of data types that are needed for its required function. It should, however, still be able to pass all data types onto the succeeding stage even

though it does not recognize them. This enables communication between non-adjacent pipeline stages.

Although not shown in Fig. 1, there are data lines, either single lines or several parallel lines, which form a data bus that also lead into and out of each pipeline stage. As is explained and illustrated in greater detail below, data is transferred into, out of, and between the stages of the pipeline over the data lines.

Note that the first pipeline stage may receive data and control signals from any form of preceding device. For example, reception circuitry of a digital image transmission system, another pipeline, or the like. On the other hand, it may generate itself, all or part of the data to be processed in the pipeline. Indeed, as is explained below, a "stage" may contain arbitrary processing circuitry, including none at all (for simple passing of data) or entire systems (for example, another pipeline or even multiple systems or pipelines), and it may generate, change, and delete data as desired.

When a pipeline stage contains valid data that is to be transferred down the pipeline, the VALID signal, which indicates data validity, need not be transferred further than to the immediately subsequent pipeline stage. A two-wire interface is, therefore, included between every pair of pipeline stages in the system. This includes a two-wire interface between a preceding device and the first stage, and between a subsequent device and the last stage, if such other devices are included and data is to be transferred between them and the pipeline.

Each of the signals, ACCEPT and VALID, has a HIGH and a LOW value. These values are abbreviated as "H" and "L", respectively. The most common applications of the pipeline, in practicing the invention, will typically be digital. In such digital implementations, the HIGH value may, for

0972666-04304

example, be a logical "1" and the LOW value may be a logical "0". The system is not restricted to digital implementations, however, and in analog implementations, the HIGH value may be a voltage or other similar quantity above (or below) a set threshold, with the LOW value being indicated by the corresponding signal being below (or above) the same or some other threshold. For digital applications, the present invention may be implemented using any known technology, such as CMOS, bipolar etc.

It is not necessary to use a distinct storage device and wires to provide for storage of VALID signals. This is true even in a digital embodiment. All that is required is that the indication of "validity" of the data be stored along with the data. By way of example only, in digital television pictures that are represented by digital values, as specified in the international standard CCIR 601, certain specific values are not allowed. In this system, eight-bit binary numbers are used to represent samples of the picture and the values zero and 255 may not be used.

If such a picture were to be processed in a pipeline built in the practice of the present invention, then one of these values (zero, for example) could be used to indicate that the data in a specific stage in the pipeline is not valid. Accordingly, any non-zero data would be deemed to be valid. In this example, there is no specific latch that can be identified and said to be storing the "validness" of the associated data. Nonetheless, the validity of the data is stored along with the data.

As shown in Fig. 1, the state of the VALID signal into each stage is indicated as an "H" or an "L" on an upper, right-pointed arrow. Therefore, the VALID signal from Stage A into Stage B is LOW, and the VALID signal from Stage D into Stage E is HIGH. The state of the ACCEPT signal into each stage is indicated as an "H" or an "L" on a lower, left-

pointing arrow. Hence, the ACCEPT signal from Stage E into Stage D is HIGH, whereas the ACCEPT signal from the device connected downstream of the pipeline into Stage F is LOW.

5 Data is transferred from one stage to another during a cycle (explained below) whenever the ACCEPT signal of the downstream stage into its upstream neighbor is HIGH. If the ACCEPT signal is LOW between two stages, then data is not transferred between these stages.

10 Referring again to Fig. 1, if a box is shaded, the corresponding pipeline stage is assumed, by way of example, to contain valid output data. Likewise, the VALID signal which is passed from that stage to the following stage is HIGH. Fig. 1 illustrates the pipeline when stages B, D, and E contain valid data. Stages A, C, and F do not contain
15 valid data. At the beginning, the VALID signal into pipeline stage A is HIGH, meaning that the data on the transmission line into the pipeline is valid.

Also at this time, the ACCEPT signal into pipeline stage F is LOW, so that no data, whether valid or not, is
20 transferred out of Stage F. Note that both valid and invalid data is transferred between pipeline stages. Invalid data, which is data not worth saving, may be written over, thereby, eliminating it from the pipeline. However, valid data must not be written over since it is data that must be saved for
25 processing or use in a downstream device e.g., a pipeline stage, a device or a system connected to the pipeline that receives data from the pipeline.

In the pipeline illustrated in Fig. 1, Stage E contains valid data D1, Stage D contains valid data D2, Stage B
30 contains valid data D3, and a device (not shown) connected to the pipeline upstream contains data D4 that is to be transferred into and processed in the pipeline. Stages B, D and E, in addition to the upstream device, contain valid data and, therefore, the VALID signal from these stages or devices

Assume now that the device connected downstream from the pipeline is not ready to accept data from the pipeline. The device signals this by setting the corresponding ACCEPT signal LOW into Stage F. Stage F itself, however, does not contain valid data and is, therefore, able to accept data from the preceding Stage E. Hence, the ACCEPT signal from Stage F into Stage E is set HIGH.

In the illustrated example, the downstream device is still not ready to accept new data in Cycle 2 and, therefore, the ACCEPT signal into Stage F is still LOW. Stage F cannot, therefore, accept new data since doing so would cause valid data D1 to be overwritten and lost. The ACCEPT signal from Stage F into Stage E, therefore, goes LOW, as does the ACCEPT signal from Stage E into Stage D since Stage E also contains valid data D2. All of the Stages A-D, however, are able to accept new data (either because they do not contain valid data or because they are able to shift their valid data downstream and accept new data) and they signal this condition to their immediately preceding neighbors by setting their corresponding ACCEPT signals HIGH.

The state of the pipelines after Cycle 2 is illustrated in Fig. 1 for the row labelled Cycle 3. By way of example, it is assumed that the downstream device is still not ready to

accept new data from Stage F (the ACCEPT signal into Stage F is LOW). Stages E and F, therefore, are still "blocked", but in Cycle 3, Stage D has received the valid data D3, which has overwritten the invalid data that was previously in this stage. Since Stage D cannot pass on data D3 in Cycle 3, it cannot accept new data and, therefore, sets the ACCEPT signal into Stage C LOW. However, stages A-C are ready to accept new data and signal this by setting their corresponding ACCEPT signals HIGH. Note that data D4 has been shifted from Stage A to Stage B.

Assume now that the downstream device becomes ready to accept new data in Cycle 4. It signals this to the pipeline by setting the ACCEPT signal into Stage F HIGH. Although Stages C-F contain valid data, they can now shift the data downstream and are, thus, able to accept new data. Since each stage is therefore able to shift data one step downstream, they set their respective ACCEPT signals out HIGH.

As long as the ACCEPT signal into the final pipeline stage (in this example, Stage F) is HIGH, the pipeline shown in Fig. 1 acts as a rigid pipeline and simply shifts data one step downstream on each cycle. Accordingly, in Cycle 5, data D1, which was contained in Stage F in Cycle 4, is shifted out of the pipeline to the subsequent device, and all other data is shifted one step downstream.

Assume now, that the ACCEPT signal into Stage F goes LOW in Cycle 5. Once again, this means that Stages D-F are not able to accept new data, and the ACCEPT signals out of these stages into their immediately preceding neighbors go LOW. Hence, the data D2, D3 and D4 cannot shift downstream, however, the data D5 can. The corresponding state of the pipeline after Cycle 5 is, thus, shown in Fig. 1 as Cycle 6.

The ability of the pipeline, in accordance with the preferred embodiments of the present invention, to "fill up"

6972698-03004
100679 9692760

empty processing stages is highly advantageous since the processing stages in the pipeline thereby become decouple from one another. In other words, even though a pipeline stage may not be ready to accept data, the entire pipeline does not have to stop and wait for the delayed stage. Rather, when one stage is unable to accept valid data it simply forms a temporary "wall" in the pipeline. Nonetheless, stages downstream of the "wall" can continue to advance valid data even to circuitry connected to the pipeline, and stages to the left of the "wall" can still accept and transfer valid data downstream. Even when several pipeline stages temporarily cannot accept new data, other stages can continue to operate normally. In particular, the pipeline can continue to accept data into its initial stage A as long as stage A does not already contain valid data that cannot be advanced due to the next stage not being ready to accept new data. As this example illustrates, data can be transferred into the pipeline and between stages even when one or more processing stages is blocked.

In the embodiment shown in Fig. 1, it is assumed that the various pipeline stages do not store the ACCEPT signals they receive from their immediately following neighbors. Instead, whenever the ACCEPT signal into a downstream stage goes LOW, this LOW signal is propagated upstream as far as the nearest pipeline stage that does not contain valid data. For example, referring to Fig. 1, it was assumed that the ACCEPT signal into Stage F goes LOW in Cycle 1. In Cycle 2, the LOW signal propagates from Stage F back to Stage D.

In Cycle 3, when the data D3 is latched into Stage D, the ACCEPT signal propagates upstream four stages to Stage C. When the ACCEPT signal into Stage F goes HIGH in Cycle 4, it must propagate upstream all the way to Stage C. In other words, the change in the ACCEPT signal must propagate back four stages. It is not necessary, however, in the embodiment

illustrated in Fig. 1, for the ACCEPT signal to propagate all the way back to the beginning of the pipeline if there is some intermediate stage that is able to accept new data.

In the embodiment illustrated in Fig. 1, each pipeline stage will still need separate input and output data latches to allow data to be transferred between stages without unintended overwriting. Also, although the pipeline illustrated in Fig. 1 is able to "compress" when downstream pipeline stages are blocked, i.e., they cannot pass on the data they contain, the pipeline does not "expand" to provide stages that contain no valid data between stages that do contain valid data. Rather, the ability to compress depends on there being cycles during which no valid data is presented to the first pipeline stage.

In Cycle 4, for example, if the ACCEPT signal into Stage F remained LOW and valid data filled pipeline stages A and B, as long as valid data continued to be presented to Stage A the pipeline would not be able to compress any further and valid input data could be lost. Nonetheless, the pipeline illustrated in Fig. 1 reduces the risk of data loss since it is able to compress as long as there is a pipeline stage that does not contain valid data.

Fig. 2 illustrates another embodiment of the pipeline that can both compress and expand in a logical manner and which includes circuitry that limits propagation of the ACCEPT signal to the nearest preceding stage. Although the circuitry for implementing this embodiment is explained and illustrated in greater detail below, Fig. 2 serves to illustrate the principle by which it operates.

For ease of comparison only, the input data and ACCEPT signals into the pipeline embodiment shown in Fig. 2 are the same as in the pipeline embodiment shown in Fig. 1. Accordingly, stages E, D and B contain valid data D1, D2 and D3, respectively. The ACCEPT signal into Stage F is LOW; and

FIG. 2

data D4 is presented to the beginning pipeline Stage A. In Fig. 2, three lines are shown connecting each neighboring pair of pipeline stages. The uppermost line, which may be a bus, is a data line. The middle line is the line over which the VALID signal is transferred, while the bottom line is the line over which the ACCEPT signal is transferred. Also, as before, the ACCEPT signal into Stage F remains LOW except in Cycle 4. Furthermore, additional data D5 is presented to the pipeline in Cycle 4.

In Fig. 2, each pipeline stage is represented as a block divided into two halves to illustrate that each stage in this embodiment of the pipeline includes primary and secondary data storage elements. In Fig. 2, the primary data storage is shown as the right half of each stage. However, it will be appreciated that this delineation is for the purpose of illustration only and is not intended as a limitation.

As Fig. 2 illustrates, as long as the ACCEPT signal into a stage is HIGH, data is transferred from the primary storage elements of the stage to the secondary storage elements of the following stage during any given cycle. Accordingly, although the ACCEPT signal into Stage F is LOW, the ACCEPT signal into all other stages is HIGH so that the data D1, D2 and D3 is shifted forward one stage in Cycle 2 and the data D4 is shifted into the first Stage A.

Up to this point, the pipeline embodiment shown in Fig. 2 acts in a manner similar to the pipeline embodiment shown in Fig. 1. The ACCEPT signal from Stage F into Stage E, however, is HIGH even though the ACCEPT signal into Stage F is LOW. As is explained below, because of the secondary storage elements, it is not necessary for the LOW ACCEPT signal to propagate upstream beyond Stage F. Moreover, by leaving the ACCEPT signal into Stage E HIGH, Stage F signals that it is ready to accept new data. Since Stage F is not able to transfer the data D1 in its primary storage elements

downstream (the ACCEPT signal into Stage F is LOW) in Cycle 3, Stage E must, therefore, transfer the data D2 into the secondary storage elements of Stage F. Since both the primary and the secondary storage elements of Stage F now contain valid data that cannot be passed on, the ACCEPT signal from Stage F into Stage E is set LOW. Accordingly, this represents a propagation of the LOW ACCEPT signal back only one stage relative to Cycle 2, whereas this ACCEPT signal had to be propagated back all the way to Stage C in the embodiment shown in Fig. 1.

Since Stages A-E are able to pass on their data, the ACCEPT signals from the stages into their immediately preceding neighbors are set HIGH. Consequently, the data D3 and D4 are shifted one stage to the right so that, in Cycle 4, they are loaded into the primary data storage elements of Stage E and Stage C, respectively. Although Stage E now contains valid data D3 in its primary storage elements, its secondary storage elements can still be used to store other data without risk of overwriting any valid data.

Assume now, as before, that the ACCEPT signal into Stage F becomes HIGH in Cycle 4. This indicates that the downstream device to which the pipeline passes data is ready to accept data from the pipeline. Stage F, however, has set its ACCEPT signal LOW and, thus, indicates to Stage E that Stage F is not prepared to accept new data. Observe that the ACCEPT signals for each cycle indicate what will "happen" in the next cycle, that is, whether data will be passed on (ACCEPT HIGH) or whether data must remain in place (ACCEPT LOW). Therefore, from Cycle 4 to Cycle 5, the data D1 is passed from Stage F to the following device, the data D2 is shifted from secondary to primary storage in Stage F, but the data D3 in Stage E is not transferred to Stage F. The data D4 and D5 can be transferred into the following pipeline stages as normal since the following stages have their ACCEPT

signals HIGH.

Comparing the state of the pipeline in Cycle 4 and Cycle 5, it can be seen that the provision of secondary storage elements, enables the pipeline embodiment shown in Fig. 2 to expand, that is, to free up data storage elements into which valid data can be advanced. For example, in Cycle 4, the data blocks D1, D2 and D3 form a "solid wall" since their data cannot be transferred until the ACCEPT signal into Stage F goes HIGH. Once this signal does become HIGH, however, data D1 is shifted out of the pipeline, data D2 is shifted into the primary storage elements of Stage F, and the secondary storage elements of Stage F become free to accept new data if the following device is not able to receive the data D2 and the pipeline must once again "compress." This is shown in Cycle 6, for which the data D3 has been shifted into the secondary storage elements of Stage F and the data D4 has been passed on from Stage D to Stage E as normal.

Figs. 3a(1), 3a(2), 3b(1) and 3b(2) (which are referred to collectively as Fig. 3) illustrate generally a preferred embodiment of the pipeline. This preferred embodiment implements the structure shown in Fig. 2 using a two-phase, non-overlapping clock with phases $\phi 0$ and $\phi 1$. Although a two-phase clock is preferred, it will be appreciated that it is also possible to drive the various embodiments of the invention using a clock with more than two phases.

As shown in Fig. 3, each pipeline stage is represented as having two separate boxes which illustrate the primary and secondary storage elements. Also, although the VALID signal and the data lines connect the various pipeline stages as before, for ease of illustration, only the ACCEPT signal is shown in Fig. 3. A change of state during a clock phase of certain of the ACCEPT signals is indicated in Fig. 3 using an upward-pointing arrow for changes from LOW to HIGH. Similarly, a downward-pointing arrow for changes from HIGH to

LOW. Transfer of data from one storage element to another is indicated by a large open arrow. It is assumed that the VALID signal out of the primary or secondary storage elements of any given stage is HIGH whenever the storage elements contain valid data.

In Fig. 3, each cycle is shown as consisting of a full period of the non-overlapping clock phases $\phi 0$ and $\phi 1$. As is explained in greater detail below, data is transferred from the secondary storage elements (shown as the left box in each stage) to the primary storage elements (shown as the right box in each stage) during clock cycle $\phi 1$, whereas data is transferred from the primary storage elements of one stage to the secondary storage elements of the following stage during the clock cycle $\phi 0$. Fig. 3 also illustrates that the primary and secondary storage elements in each stage are further connected via an internal acceptance line to pass an ACCEPT signal in the same manner that the ACCEPT signal is passed from stage to stage. In this way, the secondary storage element will know when it can pass its data to the primary storage element.

Fig. 3 shows the $\phi 1$ phase of Cycle 1, in which data D1, D2 and D3, which were previously shifted into the secondary storage elements of Stages E, D and B, respectively, are shifted into the primary storage elements of the respective stage. During the $\phi 1$ phase of Cycle 1, the pipeline, therefore, assumes the same configuration as is shown as Cycle 1 of Fig. 2. As before, the ACCEPT signal into Stage F is assumed to be LOW. As Fig. 3 illustrates, however, this means that the ACCEPT signal into the primary storage element of Stage F is LOW, but since this storage element does not contain valid data, it sets the ACCEPT signal into its secondary storage element HIGH.

The ACCEPT signal from the secondary storage elements of Stage F into the primary storage elements of Stage E is also

set HIGH since the secondary storage elements of Stage F do not contain valid data. As before, since the primary storage elements of Stage F are able to accept data, data in all the upstream primary and secondary storage elements can be shifted downstream without any valid data being overwritten. The shift of data from one stage to the next takes place during the next 00 phase in Cycle 2. For example, the valid data D1 contained in the primary storage element of Stage E is shifted into the secondary storage element of Stage F, the data D4 is shifted into the pipeline, that is, into the secondary storage element of Stage A, and so forth.

The primary storage element of Stage F still does not contain valid data during the 00 phase in Cycle 2 and, therefore, the ACCEPT signal from the primary storage elements into the secondary storage elements of Stage F remains HIGH. During the 01 phase in Cycle 2, data can therefore be shifted yet another step to the right, i.e., from the secondary to the primary storage elements within each stage.

However, once valid data is loaded into the primary storage elements of Stage F, if the ACCEPT into Stage F from the downstream device is still LOW, it is not possible to shift data out of the secondary storage element of Stage F without overwriting and destroying the valid data D1. The ACCEPT signal from the primary storage elements into the secondary storage elements of Stage F therefore goes LOW. Data D2, however, can still be shifted into the secondary storage of Stage F since it did not contain valid data and its ACCEPT signal out was HIGH.

During the 01 phase of Cycle 3, it is not possible to shift data D2 into the primary storage elements of Stage F, although data can be shifted within all the previous stages. Once valid data is loaded into the secondary storage elements of Stage F, however, Stage F is not able to pass on this

100-332480

data. It signals this event setting its ACCEPT signal out LOW.

Assuming that the ACCEPT signal into Stage F remains LOW, data upstream of Stage F can continue to be shifted between stages and within stages on the respective clock phases until the next valid data block D3 reaches the primary storage elements of Stage E. As illustrated, this condition is reached during the ϕ_1 phase of Cycle 4.

During the ϕ_0 phase of Cycle 5, data D3 has been loaded into the primary storage element of Stage E. Since this data cannot be shifted further, the ACCEPT signal out of the primary storage elements of Stage E is set LOW. Upstream data can be shifted as normal.

Assume now, as in Cycle 5 of Fig. 2, that the device connected downstream of the pipeline is able to accept pipeline data. It signals this event by setting the ACCEPT signal into pipeline Stage F HIGH during the ϕ_1 phase of Cycle 4. The primary storage elements of Stage F can now shift data to the right and they are also able to accept new data. Hence, the data D1 was shifted out during the ϕ_1 phase of Cycle 5 so that the primary storage elements of Stage F no longer contain data that must be saved. During the ϕ_1 phase of Cycle 5, the data D2 is, therefore, shifted within Stage F from the secondary storage elements to the primary storage elements. The secondary storage elements of Stage F are also able to accept new data and signal this by setting the ACCEPT signal into the primary storage elements of Stage E HIGH. During transfer of data within a stage, that is, from its secondary to its primary storage elements, both sets of storage elements will contain the same data, but the data in the secondary storage elements can be overwritten with no data loss since this data will also be held in the primary storage elements. The same holds true for data transfer from the primary storage elements of one stage into the secondary

09772698 043004

storage elements of a subsequent stage.

Assume now, that the ACCEPT signal into the primary storage elements of Stage F goes LOW during the $\phi 1$ phase in Cycle 5. This means that Stage F is not able to transfer the data D2 out of the pipeline. Stage F, consequently, sets the ACCEPT signal from its primary to its secondary storage elements LOW to prevent overwriting of the valid data D2. The data D2 stored in the secondary storage elements of Stage F, however, can be overwritten without loss, and the data D3, is therefore, transferred into the secondary storage elements of Stage F during the $\phi 0$ phase of Cycle 6. Data D4 and D5 can be shifted downstream as normal. Once valid data D3 is stored in Stage F along with data D2, as long as the ACCEPT signal into the primary storage elements of Stage F is LOW, neither of the secondary storage elements can accept new data, and it signals this by setting the ACCEPT signal into Stage E LOW.

When the ACCEPT signal into the pipeline from the downstream device changes from LOW to HIGH or vice versa, this change does not have to propagate upstream within the pipeline further than to the immediately preceding storage elements (within the same stage or within the preceding pipeline stage). Rather, this change propagates upstream within the pipeline one storage element block per clock phase.

As this example illustrates, the concept of a "stage" in the pipeline structure illustrated in Fig. 3 is to some extent a matter of perception. Since data is transferred within a stage (from the secondary to the primary storage elements) as it is between stages (from the primary storage elements of the upstream stage into the secondary storage elements of the neighboring downstream stage), one could just as well consider a stage to consist of "primary" storage elements followed by "secondary storage elements" instead of

5

10

20

22

elements. The VALID signal may also be processed in an analogous manner.

A great advantage of the two-wire interface (one wire for each of the VALID and ACCEPT signals) is its ability to control the pipeline without the control signals needing to propagate back up the pipeline all the way to its beginning stage. Referring once again to Fig. 1, Cycle 3, for example, although stage F "tells" stage E that it cannot accept data, and stage E tells stage D, and stage D tells stage C. Indeed, if there had been more stages containing valid data, then this signal would have propagated back even further along the pipeline. In the embodiment shown in Fig. 3, Cycle 3, the LOW ACCEPT signal is not propagated any further upstream than to Stage E and, then, only to its primary storage elements.

As described below, this embodiment is able to achieve this flexibility without adding significantly to the silicon area that is required to implement the design. Typically, each latch in the pipeline used for data storage requires only a single extra transistor (which lays out very efficiently in silicon). In addition, two extra latches and a small number of gates are preferably added to process the ACCEPT and VALID signals that are associated with the data latches in each half-stage.

Fig. 4 illustrates a hardware structure that implements a stage as shown in Fig. 3.

By way of example only, it is assumed that eight-bit data is to be transferred (with or without further manipulation in optional combinatorial logic circuits) in parallel through the pipeline. However, it will be appreciated that either more or less than eight-bit data can be used in practicing the invention. Furthermore, the two-wire interface in accordance with this embodiment is, however, suitable for use with any data bus width, and the data bus width may even

change from one stage to the next if a particular application so requires. The interface in accordance with this embodiment can also be used to process analog signals.

As discussed previously, while other conventional timing arrangements may be used, the interface is preferably controlled by a two-phase, non-overlapping clock. In Figs. 4-9, these clock phase signals are referred to as PH0 and PH1. In Fig. 4, a line is shown for each clock phase signal.

Input data enters a pipeline stage over a multi-bit data bus IN_DATA and is transferred to a following pipeline stage or to subsequent receiving circuitry over an output data bus OUT_DATA. The input data is first loaded in a manner described below into a series of input latches (one for each input data signal) collectively referred to as LDIN, which constitute the secondary storage elements described above.

In the illustrated example of this embodiment, it is assumed that the Q outputs of all latches follow their D inputs, that is, they are "loaded", when the clock input is HIGH, i.e., at a logic "1" level. Additionally, the Q outputs hold their last values. In other words, the Q outputs are "latched" on the falling edge of their respective clock signals. Each latch has for its clock either one of two non-overlapping clock signals PH0 or PH1 (as shown in Fig. 5), or the logical AND combination of one of these clock signals PH0, PH1 and one logic signal. The invention works equally well, however, by providing latches that latch on the rising edges of the clock signals, or any other known latching arrangement, as long as conventional methods are applied to ensure proper timing of the latching operations.

The output data from the input data latch LDIN passes via an arbitrary and optional combinatorial logic circuit B1, which may be provided to convert output data from input latch LDIN into intermediate data, which is then later loaded in an output data latch LDOUT, which comprises the primary storage

100-000000-000000

elements described above. The output from the output data latch LDOUT may similarly pass through an arbitrary and optional combinatorial logic circuit B2 before being passed onward as OUT_DATA to the next device downstream. This may
 5 be another pipeline stage or any other device connected to the pipeline.

In the practice of the present invention, each stage of the pipeline also includes a validation input latch LVIN, a validation output latch LVOUT, an acceptance input latch
 10 LAIN, and an acceptance output latch LAOUT. Each of these four latches is, preferably, a simple, single-stage latch. The outputs from latches LVIN, LVOUT, LAIN and LAOUT are, respectively, QVIN, QVOUT, QAIN, QAOUT. The output signal QVIN from the validation input latch is connected either
 15 directly as an input to the validation output latch LVOUT, or via intermediate logic devices or circuits that may alter the signal.

Similarly, the output validation signal QVOUT of a given stage may be connected either directly to the input of the validation input latch QVIN of the following stage, or via
 20 intermediate devices or logic circuits, which may alter the validation signal. This output QVIN is also connected to a logic gate (to be described below), whose output is connected to the input of the acceptance input latch LAIN. The output QAOUT from the acceptance output latch LAOUT is connected to
 25 a similar logic gate (described below), optionally via another logic gate.

As shown in Fig. 4, the output validation signal QVOUT forms an OUT_VALID signal that can be received by subsequent
 30 stages as an IN_VALID signal, or simply to indicate valid data to subsequent circuitry connected to the pipeline. The readiness of the following circuit or stage to accept data is indicated to each stage as the signal OUT_ACCEPT, which is connected as the input to the acceptance output latch LAOUT,

preferably via logic circuitry, which is described below. Similarly, the output QAOUT of the acceptance output latch LAOUT is connected as the input to the acceptance input latch LAIN, preferably via logic circuitry, which is described below.

In practicing the present invention, the output signals QVIN, QVOUT from the validation latches LVIN, LVOUT are combined with the acceptance signals QAOUT, OUT_ACCEPT, respectively, to form the inputs to the acceptance latches LAIN, LAOUT, respectively. In the embodiment illustrated in Fig. 4, these input signals are formed as the logical NAND combination of the respective validation signals QVIN, QVOUT, with the logical inverse of the respective acceptance output signals QAOUT, OUT_ACCEPT. Conventional logic gates, NAND1 and NAND2, perform the NAND operation, and the inverters INV1, INV2 form the logical inverses of the respective acceptance signals.

As is well known in the art of digital design, the output from a NAND gate is a logical "1" when any or all of its input signals are in the logical "0" state. The output from a NAND gate is, therefore, a logical "0" only when all of its inputs are in the logical "1" state. Also well known in the art, is that the output of a digital inverter such as INV1 is a logical "1" when its input signal is a "0" and is a "0" when its input signal is a "1"

The inputs to the NAND gate NAND1 are, therefore, QVIN and NOT (QAOUT), where "NOT" indicates binary inversion. Using known techniques, the input to the acceptance latch LAIN can be resolved as follows:

$$\text{NAND}(\text{QVIN}, \text{NOT}(\text{QAOUT})) = \text{NOT}(\text{QVIN}) \text{ OR } \text{QAOUT}$$

In other words, the combination of the inverter INV1 and the NAND gate NAND1 is a logical "1" either when the signal QVIN is a "0" or the signal QAOUT is a "1", or both. The gate NAND1 and the inverter INV1 can, therefore, be

5 As is well known in the art of digital design, many
latches suitable for use as the validation and acceptance
latches may have two outputs, Q and NOT(Q), that is, Q and
its logical inverse. If such latches are chosen, the one
input to the OR gate can, therefore, be tied directly to the
10 NOT(Q) output of the validation latch LVIN. The gate NAND1
and the inverter INV1 can be implemented using well known
conventional techniques. Depending on the latch architecture
used, however, it may be more efficient to use a latch
without an inverting output, and to provide instead the gate
15 NAND1 and the inverter INV1, both of which also can be
implemented efficiently in a silicon device. Accordingly,
any known arrangement may be used to generate the Q signal
and/or its logical inverse.

The data and validation latches LDIN, LDOUT, LVIN and LVOUT, load their respective data inputs when both clock signals (PH0 at the input side and PH1 at the output side) and the output from the acceptance latch of the same side are logical "1". Thus, the clock signal (PH0 for the input latches LDIN and LVIN) and the output of the respective acceptance latch (in this case, LAIN) are used in a logical AND manner and data is loaded only when they are both logical "1".

In particular applications, such as CMOS implementations of the latches, the logical AND operation that controls the loading (via the illustrated CK or enabling "input") of the latches can be implemented easily in a conventional manner by connecting the respective enabling input signals (for example, PH0 and QAIN for the latches LVIN and LDIN), to the gates of MOS transistors connected in series in the input

lines of the latches. Consequently, is necessary to provide an actual logic AND gate, which might cause problems of timing due to propagation delay in high-speed applications. The AND gate shown in the figures, therefore, only indicates
 5 the logical function to be performed in generating the enable signals of the various latches.

Thus, the data latch LDIN loads input data only when PH0 and QAIN are both "1". It will latch this data when either of these two signals goes to a "0".

10 Although only one of the clock phase signals PH0 or PH1, is used to clock the data and validation latches at the input (and output) side of the pipeline stage, the other clock phase signal is used, directly, to clock the acceptance latch at the same side. In other words, the acceptance latch on
 15 either side (input or output) of a pipeline stage is preferably clocked "out of phase" with the data and validation latches on the same side. For example, PH1 is used to clock the acceptance input latch, although PH0 is used in generating the clock signal CK for the data latch
 20 LDIN and the validation latch LVIN.

As an example of the operation of a pipeline augmented by the two-wire validation and acceptance circuitry assume that no valid data is initially presented at the input to the circuit, either from a preceding pipeline stage, or from a
 25 transmission device. In other words, assume that the validation input signal IN_VALID to the illustrated stage has not gone to a "1" since the system was most recently reset. Assume further that several clock cycles have taken place since the system was last reset and, accordingly, the
 30 circuitry has reached a steady-state condition. The validation input signal QVIN from the validation latch LVIN is, therefore, loaded as a "0" during the next positive period of the clock PH0. The input to the acceptance input latch LAIN (via the gate NAND1 or another equivalent gate),

is, therefore, loaded as a "1" during the next positive period of the clock signal PH1. In other words, since the data in the data input latch LDIN is not valid, the stage signals that it is ready to accept input data (since it does not hold any data worth saving).

In this example, note that the signal IN_ACCEPT is used to enable the data and validation latches LDIN and LVIN. Since the signal IN_ACCEPT at this time is a "1", these latches effectively work as conventional transparent latches so that whatever data is on the IN_DATA bus simply is loaded into the data latch LDIN as soon as the clock signal PH0 goes to a "1". Of course, this invalid data will also be loaded into the next data latch LDOUT of the following pipeline stage as long as the output QAOUT from its acceptance latch is a "1".

Hence, as long as a data latch does not contain valid data, it accepts or "loads" any data presented to it during the next positive period of its respective clock signal. On the other hand, such invalid data is not loaded in any stage for which the acceptance signal from its corresponding acceptance latch is low (that is, a "0"). Furthermore, the output signal from a validation latch (which forms the validation input signal to the subsequent validation latch) remains a "0" as long as the corresponding IN_VALID (or QVIN) signal to the validation latch is low.

When the input data to a data latch is valid, the validation signal IN_VALID indicates this by rising to a "1". The output of the corresponding validation latch then rises to a "1" on the next rising edge of its respective clock phase signal. For example, the validation input signal QVIN of latch LVIN rises to a "1" when its corresponding IN_VALID signal goes high (that is, rises to a "1") on the next rising edge of the clock phase signal PH0.

Assume now, instead, that the data input latch LDIN contains valid data. If the data output latch LDOUT is read,

to accept new data, its acceptance signal QAOUT will be a "1". In this case, during the next positive period of the clock signal PH1, the data latch LDOUT and validation latch LVOUT will be enabled, and the data latch LDOUT will load the data present at its input. This will occur before the next rising edge of the other clock signal PH0, since the clock signals are non-overlapping. At the next rising edge of PH0, the preceding data latch (LDIN) will, therefore, not latch in new input data from the preceding stage until the data output latch LDOUT has safely latched the data transferred from the latch LDIN.

Accordingly, the same sequence is followed by every adjacent pair of data latches (within a stage or between adjacent stages) that are able to accept data, since they will be operating based on alternate phases of the clock. Any data latch that is not ready to accept new data because it contains valid data that cannot yet be passed, will have an output acceptance signal (the QA output from its acceptance latch LA) that is LOW, and its data latch LDIN or LDOUT will not be loaded. Hence, as long as the acceptance signal (the output from the acceptance latch) of a given stage or side (input or output) of a stage is LOW, its corresponding data latch will not be loaded.

Fig. 4 also shows a reset feature included in a preferred embodiment. In the illustrated example, a reset signal NOTRESET0 is connected to an inverting reset input R (inversion is hereby indicated by a small circle, as is conventional) of the validation output latch LVOUT. As is well known, this means that the validation latch LVOUT will be forced to output a "0" whenever the reset signal NOTRESET0 becomes a "0". One advantage of resetting the latch when the reset signal goes low (becomes a "0") is that a break in transmission will reset the latches. They will then be in their "null" or reset state whenever a valid transmission

begins and the reset signal goes HIGH. The reset signal NOTRESET0, therefore, operates as a digital "ON/OFF" switch, such that it must be at a HIGH value in order to activate the pipeline.

5 Note that it is not necessary to reset all of the latches that hold valid data in the pipeline. As depicted in Fig. 4, the validation input latch LVIN is not directly reset by the reset signal NOTRESET0, but rather is reset indirectly. Assume that the reset signal NOTRESET0 drops to a "0". The
10 validation output signal QVOUT also drops to a "0", regardless of its previous state, whereupon the input to the acceptance output latch LAOUT (via the gate NAND1) goes HIGH. The acceptance output signal QAOUT also rises to a "1". This QAOUT value of "1" is then transferred as a "1" to the input
15 of the acceptance input latch LAIN regardless of the state of the validation input signal QVIN. The acceptance input signal QAIN then rises to a "1" at the next rising edge of the clock signal PH1. Assuming that the validation signal IN_VALID has been correctly reset to a "0", then upon the
20 subsequent rising edge of the clock signal PH0, the output from the validation latch LVIN will become a "0", as it would have done if it had been reset directly.

As this example illustrates, it is only necessary to reset the validation latch in only one side of each stage
25 (including the final stage) in order to reset all validation latches. In fact, in many applications, it will not be necessary to reset every other validation latch: If the reset signal NOTRESET0 can be guaranteed to be low during more than one complete cycle of both phases PH0, PH1 of the
30 clock, then the "automatic reset" (a backwards propagation of the reset signal) will occur for validation latches in preceding pipeline stages. Indeed, if the reset signal is held low for at least as many full cycles of both phases of the clock as there are pipeline stages, it will only re

necessary to directly reset the validation output latch in the final pipeline stage.

Figs. 5a and 5b (referred to collectively as Fig. 5) illustrate a timing diagram showing the relationship between the non-overlapping clock signals PH0, PH1, the effect of the reset signal, and the holding and transfer of data for the different permutations of validation and acceptance signals into and between the two illustrated sides of a pipeline stage configured in the embodiment shown in Fig. 4. In the example illustrated in the timing diagram of Fig. 5, it has been assumed that the outputs from the data latches LDIN, LDOUT are passed without further manipulation by intervening logic blocks B1, B2. This is by way of example and not necessarily by way of limitation. It is to be understood that any combinatorial logic structures may be included between the data latches of consecutive pipeline stages, or between the input and output sides of a single pipeline stage. The actual illustrated values for the input data (for example the HEX data words "aa" or "04") are also merely illustrative. As is mentioned above, the input data bus may have any width (and may even be analog), as long as the data latches or other storage devices are able to accommodate and latch or store each bit or value of the input word.

Preferred Data Structure - "tokens"

In the sample application shown in Fig. 4, each stage processes all input data, since there is no control circuitry that excludes any stage from allowing input data to pass through its combinatorial logic block B1, B2, and so forth. To provide greater flexibility, the present invention includes a data structure in which "tokens" are used to distribute data and control information throughout the system. Each token consists of a series of binary bits separated into one or more blocks of token words.

Furthermore, the bits fall into one of three types: address bits (A), data bits (D), or an extension bit (E). Assume by way of example and, not necessarily by way of limitation, that data is transferred as words over an 8-bit bus with a 1-bit extension bit line. An example of a four-word token is, in order of transmission:

| | | | | | | | | | |
|--------------|---|---|---|---|---|---|---|---|---|
| First word: | E | A | A | A | D | D | D | D | D |
| Second word: | E | D | D | D | D | D | D | D | D |
| Third word: | E | D | D | D | D | D | D | D | D |
| Fourth word: | E | D | D | D | D | D | D | D | D |

Note that the extension bit E is used as an addition (preferably) to each data word. In addition, the address field can be of variable length and is preferably transmitted just after the extension bit of the first word.

Tokens, therefore, consist of one or more words of (binary) digital data in the present invention. Each of these words is transferred in sequence and preferably in parallel, although this method of transfer is not necessary: serial data transfer is also possible using known techniques. For example, in a video parser, control information is transmitted in parallel, whereas data is transmitted serially.

As the example illustrates, each token has, preferably at the start, an address field (the string of A-bits) that identifies the type of data that is contained in the token. In most applications, a single word or portion of a word is sufficient to transfer the entire address field, but this is not necessary in accordance with the invention, so long as logic circuitry is included in the corresponding pipeline stages that is able to store some representation of partial address fields long enough for the stages to receive and decode the entire address field.

Note that no dedicated wires or registers are required to transmit the address field. It is transmitted using the data bits. As is explained below, a pipeline stage will not be slowed down if it is not intended to be activated by the particular address field, i.e., the stage will be able to pass along the token without delay.

The remainder of the data in the token following the address field is not constrained by the use of tokens. These D-data bits may take on any values and the meaning attached to these bits is of no importance here. That is, the meaning of the data can vary, for example, depending upon where the data is positioned within the system at a particular point in time. The number of data bits D appended after the address field can be as long or as short as required, and the number of data words in different tokens may vary greatly. The address field and extension bit are used to convey control signals to the pipeline stages. Because the number of words in the data field (the string of D bits) can be arbitrary, as can be the information conveyed in the data field can also vary accordingly. The explanation below is, therefore, directed to the use of the address and extension bits.

In the present invention, tokens are a particularly useful data structure when a number of blocks of circuitry are connected together in a relatively simple configuration. The simplest configuration is a pipeline of processing steps. For example, in the one shown in Fig. 1. The use of tokens, however, is not restricted to use on a pipeline structure.

Assume once again that each box represents a complete pipeline stage. In the pipeline of Fig. 1, data flows from left to right in the diagram. Data enters the machine and passes into processing Stage A. This may or may not modify the data and it then passes the data to Stage B. The modification, if any, may be arbitrarily complicated and, in general, there will not be the same number of data items

flowing into any stage as flow out. Stage B modifies the data again and passes it onto Stage C, and so forth. In a scheme such as this, it is impossible for data to flow in the opposite direction, so that, for example, Stage C cannot pass data to Stage A. This restriction is often perfectly acceptable.

On the other hand, it is very desirable for Stage A to be able to communicate information to Stage C even though there is no direct connection between the two blocks. Stage A and C communication is only via Stage B. One advantage of the tokens is their ability to achieve this kind of communication. Since any processing stage that does not recognize a token simply passes it on unaltered to the next block.

According to this example, an extension bit is transmitted along with the address and data fields in each token so that a processing stage can pass on a token (which can be of arbitrary length) without having to decode its address at all. According to this example, any token in which the extension bit is HIGH (a "1") is followed by a subsequent word which is part of the same token. This word also has an extension bit, which indicates whether there is a further token word in the token. When a stage encounters a token word whose extension bit is LOW (a "0"), it is known to be the last word of the token. The next word is then assumed to be the first word of a new token.

Note that although the simple pipeline of processing stages is particularly useful, it will be appreciated that tokens may be applied to more complicated configurations of processing elements. An example of a more complicated processing element is described below.

It is not necessary, in accordance with the present invention, to use the state of the extension bit to signal the last word of a given token by giving it an extension bit

set to "0". One alternative to the preferred scheme is to move the extension bit so that it indicates the first word of a token instead of the last. This can be accomplished with appropriate changes in the decoding hardware.

5 The advantage of using the extension bit of the present invention to signal the last word in a token rather than the first, is that it is often useful to modify the behavior of a block of circuitry depending upon whether or not a token has extension bits. An example of this is a token that
10 activates a stage that processes video quantization values stored in a quantization table (typically a memory device). For example, a table containing 64 eight-bit arbitrary binary integers.

15 In order to load a new quantization table into the quantizer stage of the pipeline, a "QUANT_TABLE" token is sent to the quantizer. In such a case the token, for example, consists of 65 token words. The first word contains the code "QUANT_TABLE", i.e., build a quantization table. This is followed by 64 words, which are the integers of the
20 quantization table.

25 When encoding video data, it is occasionally necessary to transmit such a quantization table. In order to accomplish this function, a QUANT_TABLE token with no extension words can be sent to the quantizer stage. On seeing this token, and noting that the extension bit of its first word is LOW,
30 the quantizer stage can read out its quantization table and construct a QUANT_TABLE token which includes the 64 quantization table values. The extension bit of the first word (which was LOW) is changed so that it is HIGH and the token continues, with HIGH extension bits, until the new end of the token, indicated by a LOW extension bit on the sixty fourth quantization table value. This proceeds in the typical way through the system and is encoded into the bit stream.

Continuing with the example, the quantizer may either load a new quantization table into its own memory device or read out its table depending on whether the first word of the QUANT_TABLE token has its extension bit set or not.

5 The choice of whether to use the extension bit to signal the first or last token word in a token will, therefore, depend on the system in which the pipeline will be used. Both alternatives are possible in accordance with the invention.

10 Another alternative to the preferred extension bit scheme is to include a length count at the start of the token. Such an arrangement may, for example, be efficient if a token is very long. For example, assume that a typical token in a given application is 1000 words long. Using the illustrated
15 extension bit scheme (with the bit attached to each token word), the token would require 1000 additional bits to contain all the extension bits. However, only ten bits would be required to encode the token length in binary form.

20 Although there are, therefore, uses for long tokens, experience has shown that there are many uses for short tokens. Here the preferred extension bit scheme is advantageous. If a token is only one word long, then only one bit is required to signal this. However, a counting scheme would typically require the same ten bits as before.

25 Disadvantages of a length count scheme include the following: 1) it is inefficient for short tokens; 2) it places a maximum length restriction on a token (with only ten bits, no more than 1023 words can be counted); 3) the length of a token must be known in advance of generating the count
30 (which is presumably at the start of the token); 4) every block of circuitry that deals with tokens would need to be provided with hardware to count words; and 5) if the count should get corrupted (due to a data transmission error) it is not clear whether recovery can be achieved.

The advantages of the extension bit scheme in accordance with the present invention include: 1) pipeline stages need not include a block of circuitry that decodes every token since unrecognized tokens can be passed on correctly by considering only the extension bit; 2) the coding of the extension bit is identical for all tokens; 3) there is no limit placed on the length of a token; 4) the scheme is efficient (in terms of overhead to represent the length of the token) for short tokens; and 5) error recovery is naturally achieved. If an extension bit is corrupted then one random token will be generated (for an extension bit corrupted from "1" to "0") or a token will be lost (extension bit corrupted "0" to "1"). Furthermore, the problem is localized to the tokens concerned. After that token, correct operation is resumed automatically.

In addition, the length of the address field may be varied. This is highly advantageous since it allows the most common tokens to be squeezed into the minimum number of words. This, in turn, is of great importance in video data pipeline systems since it ensures that all processing stages can be continuously running at full bandwidth.

In accordance to the present invention, in order to allow variable length address fields, the addresses are chosen so that a short address followed by random data can never be confused with a longer address. The preferred technique for encoding the address field (which also serves as the "code" for activating an intended pipeline stage) is the well-known technique first described by Huffman, hence the common name "Huffman Code". Nevertheless, it will be appreciated by one of ordinary skill in the art, that other coding schemes may also be successfully employed.

Although Huffman encoding is well understood in the field of digital design, the following example provides a general background:

5

10

20

25

二、

will be able to deal with the new token without modification to their designs because they will not recognize it and will, accordingly, pass that token on unmodified.

This ability of the present invention to leave
 5 substantially existing designed devices unaffected has clear advantages. It may be possible to leave some semiconductor chips in a chip set completely unaffected by a design improvement in some other chips in the set. This is advantageous both from the perspective of a customer and from
 10 that of a chip manufacturer. Even if modifications mean that all chips are affected by the design change (a situation that becomes increasingly likely as levels of integration progress so that the number of chips in a system drops) there will still be the considerable advantage of better time-to-market
 15 than can be achieved, since the same design can be reused.

In particular, note the situation that occurs when it becomes necessary to extend the token set to include two word addresses. Even in this case, it is still not necessary to modify an existing design. Token decoders in the pipeline
 20 stages will attempt to decode the first word of such a token and will conclude that it does not recognize the token. It will then pass on the token unmodified using the extension bit to perform this operation correctly. It will not attempt to decode the second word of the token (even though this
 25 contains address bits) because it will "assume" that the second word is part of the data field of a token that it does not recognize.

In many cases, a pipeline stage or a connected block of circuitry will modify a token. This usually, but not
 30 necessarily, takes the form of modifying the data field of a token. In addition, it is common for the number of data words in the token to be modified, either by removing certain data words or by adding new ones. In some cases, tokens are removed entirely from the token stream.

In most applications, pipeline stages will typically only decode (be activated by) a few tokens; the stage does not recognize other tokens and passes them on unaltered. In a large number of cases, only one token is decoded, the DATA Token word itself.

In many applications, the operation of a particular stage will depend upon the results of its own past operations. The "state" of the stage, thus, depends on its previous states. In other words, the stage depends upon stored state information, which is another way of saying it must retain some information about its own history one or more clock cycles ago. The present invention is well-suited for use in pipelines that include such "state machine" stages, as well as for use in applications in which the latches in the data path are simple pipeline latches.

The suitability of the two-wire interface, in accordance with the present invention, for such "state machine" circuits is a significant advantage of the invention. This is especially true where a data path is being controlled by a state machine. In this case, the two-wire interface technique above-described may be used to ensure that the "current state" of the machine stays in step with the data which it is controlling in the pipeline.

Fig. 6 shows a simplified block diagram of one example of circuitry included in a pipeline stage for decoding a token address field. This illustrates a pipeline stage that has the characteristics of a "state machine". Each word of a token includes an "extension bit" which is HIGH if there are more words in the token or LOW if this is the last word of the token. If this is the last word of a token, the next valid data word is the start of a new token and, therefore, its address must be decoded. The decision as to whether or not to decode the token address in any given word, thus, depends upon knowing the value of the previous extension bit.

For the sake of simplicity only, the two-wire interface (with the acceptance and validation signals and latches) is not illustrated and all details dealing with resetting the circuit are omitted. As before, an 8-bit data word is
 5 assumed by way of example only and not by way of limitation.

This exemplifying pipeline stage delays the data bits and the extension bit by one pipeline stage. It also decodes the DATA Token. At the point when the first word of the DATA Token is presented at the output of the circuit, the signal
 10 "DATA_ADDR" is created and set HIGH. The data bits are delayed by the latches LDIN and LDOUT, each of which is repeated eight times for the eight data bits used in this example (corresponding to an 8-input, 8-output latch). Similarly, the extension bit is delayed by extension bit
 15 latches LEIN and LEOUT.

In this example, the latch LEPREV is provided to store the most recent state of the extension bit. The value of the extension bit is loaded into LEIN and is then loaded into LEOUT on the next rising edge of the non-overlapping clock
 20 phase signal PH1. Latch LEOUT, thus, contains the value of the current extension bit, but only during the second half of the non-overlapping, two-phase clock. Latch LEPREV, however, loads this extension bit value on the next rising edge of the clock signal PH0, that is, the same signal that enables the
 25 extension bit input latch LEIN. The output QEPREV of the latch LEPREV, thus, will hold the value of the extension bit during the previous PH0 clock phase.

The five bits of the data word output from the inverting Q output, plus the non-inverted MD[2], of the latch LDIN are
 30 combined with the previous extension bit value QEPREV in a series of logic gates NAND1, NAND2, and NOR1, whose operations are well known in the art of digital design. The designation "N_MD[m] indicates the logical inverse of bit m of the mid-data word MD[7:0]. Using known techniques of

Boolean algebra, it can be shown that the output signal SA from this logic block (the output from NOR1) is HIGH (a "1") only when the previous extension bit is a "0" (QPREV="0") and the data word at the output of the non-inverting Q latch (the original input word) LDIN has the structure "000001xx", that is, the five high-order bits MD[7]-MD[3] bits are all "0" and the bit MD[2] is a "1" and the bits in the Zero-one positions have any arbitrary value.

There are, thus, four possible data words (there are four permutations of "xx") that will cause SA and, therefore, the output of the address signal latch LADDR to whose input SA is connected, to become HIGH. In other words, this stage provides an activation signal (DATA_ADDR = "1") only when one of the four possible proper tokens is presented and only when the previous extension bit was a zero, that is, the previous data word was the last word in the previous series of token words, which means that the current token word is the first one in the current token.

When the signal QPREV from latch LEPREV is LOW, the value at the output of the latch LDIN is therefore the first word of a new token. The gates NAND1, NAND2 and NOR1 decode the DATA token (000001xx). This address decoding signal SA is, however, delayed in latch LADDR so that the signal DATA_ADDR has the same timing as the output data OUT_DATA and OUT_EXTN.

Fig. 7 is another simple example of a state-dependent pipeline stage in accordance with the present invention, which generates the signal LAST_OUT_EXTN to indicate the value of the previous output extension bit OUT_EXTN. One of the two enabling signals (at the CK inputs) to the present and last extension bit latches, LEOUT and LEPREV, respectively, is derived from the gate AND1 such that these latches only load a new value for them when the data is valid and is being accepted (the Q outputs are HIGH from the output validation and acceptance latches LVOUT and LAOUT,

5

10

20

25

30

Third, rules and hardware structures for the handling of unrecognized tokens (that is, for passing them on unmodified) allow communication between one stage and a downstream stage that is not its nearest neighbor in the pipeline. This also

increases the expandability and efficient adaptability of the pipeline since it allows for future changes in the token set without requiring large scale redesigning of existing pipeline stages. The tokens of the present invention are particularly useful when used in conjunction with the two-wire interface that is described above and below.

As an example of the above, Figs. 8a and 8b, taken together (and referred to collectively below as Fig. 8), depict a block diagram of a pipeline stage whose function is as follows. If the stage is processing a predetermined token (known in this example as the DATA token), then it will duplicate every word in this token with the exception of the first one, which includes the address field of the DATA token. If, on the other hand, the stage is processing any other kind of token, it will delete every word. The overall effect is that, at the output, only DATA Tokens appear and each word within these tokens is repeated twice.

Many of the components of this illustrated system may be the same as those described in the much simpler structures shown in Figs. 4, 6, and 7. This illustrates a significant advantage. More complicated pipeline stages will still enjoy the same benefits of flexibility and elasticity, since the same two-wire interface may be used with little or no adaptation.

The data duplication stage shown in Fig. 8 is merely one example of the endless number of different types of operations that a pipeline stage could perform in any given application. This "duplication stage" illustrates, however, a stage that can form a "bottleneck", so that the pipeline according to this embodiment will "pack together".

A "bottleneck" can be any stage that either takes a relatively long time to perform its operations, or that creates more data in the pipeline than it receives. This example also illustrates that the two-wire accept/valid

interface according to this embodiment can be adapted very easily to different applications.

The duplication stage shown in Fig. 8 also has two latches LEIN and LEOUT that, as in the example shown in Fig. 6, latch
 5 the state of the extension bit at the input and at the output of the stage, respectively. As Fig. 8a shows, the input extension latch LEIN is clocked synchronously with the input data latch LDIN and the validation signal IN_VALID.

For ease of reference, the various latches included in the
 10 duplication stage are paired below with their respective output signals:

In the duplication stage, the output from the data latch LDIN forms intermediate data referred to as MID_DATA. This intermediate data word is loaded into the data output latch
 15 LDOUT only when an intermediate acceptance signal (labeled "MID_ACCEPT" in Fig. 8a) is set HIGH.

The portion of the circuitry shown in Fig. 8 below the acceptance latches LAIN, LAOUT, shows the circuits that are added to the basic pipeline structure to generate the various

FIG. 8a

internal control signals used to duplicate data. These include a "DATA_TOKEN" signal that indicates that the circuitry is currently processing a valid DATA Token, and a NOT_DUPLICATE signal which is used to control duplication of data. When the circuitry is processing a DATA Token, the NOT_DUPLICATE signal toggles between a HIGH and a LOW state and this causes each word in the token to be duplicated once (but no more times). When the circuitry is not processing a valid DATA Token then the NOT_DUPLICATE signal is held in a HIGH state. Accordingly, this means that the token words that are being processed are not duplicated.

As Fig. 8a illustrates, the upper six bits of 8-bit intermediate data word and the output signal QI1 from the latch LI1 form inputs to a group of logic gates NOR1, NOR2, NAND18. The output signal from the gate NAND18 is labeled S1. Using well-known Boolean algebra, it can be shown that the signal S1 is a "0" only when the output signal QI1 is a "1" and the MID_DATA word has the following structure: "000001xx", that is, the upper five bits are all "0", the bit MID_DATA[2] is a "1" and the bits in the MID_DATA[1] and MID_DATA[0] positions have any arbitrary value. Signal S1, therefore, acts as a "token identification signal" which is low only when the MID_DATA signal has a predetermined structure and the output from the latch LI1 is a "1". The nature of the latch LI1 and its output QI1 is explained further below.

Latch LO1 performs the function of latching the last value of the intermediate extension bit (labeled "MID_EXTN" and as signal S4), and it loads this value on the next rising edge of the clock phase PHO into the latch LI1, whose output is the bit QI1 and is one of the inputs to the token decoding logic group that forms signal S1. Signal S1, as is explained above, may only drop to a "0" if the signal QI1 is a "1" (and the MID_DATA signal has the predetermined structure). Signal

S1 may, therefore, only drop to a "0" whenever the last extension bit was "0", indicating that the previous token has ended. Therefore, the MID_DATA word is the first data word in a new token.

5 The latches LO2 and LI2 together with the NAND gates NAND20 and NAND22 form storage for the signal, DATA_TOKEN. In the normal situation, the signal QI1 at the input to NAND20 and the signal S1 at the input to NAND22 will both be at logic "1". It can be shown, again by the techniques of
10 Boolean algebra, that in this situation these NAND gates operate in the same manner as inverters, that is, the signal QI2 from the output of latch LI2 is inverted in NAND20 and then this signal is inverted again by NAND22 to form the signal S2. In this case, since there are two logical
15 inversions in this path, the signal S2 will have the same value as QI2.

It can also be seen that the signal DATA_TOKEN at the output of latch LO2 forms the input to latch LI2. As a result, as long as the situation remains in which both QI1
20 and S1 are HIGH, the signal DATA_TOKEN will retain its state (whether "0" or "1"). This is true even though the clock signals PH0 and PH1 are clocking the latches (LI2 and LO2 respectively). The value of DATA_TOKEN can only change when one or both of the signals QI1 and S1 are "0".

25 As explained earlier, the signal QI1 will be "0" when the previous extension bit was "0". Thus, it will be "0" whenever the MID_DATA value is the first word of a token (and, thus, includes the address field for the token). In this situation, the signal S1 may be either "0" or "1". As
30 explained earlier, signal S1 will be "0" if the MID_DATA word has the predetermined structure that in this example indicates a "DATA" Token. If the MID_DATA word has any other structure, (indicating that the token is some other token, not a DATA Token), S1 will be "1".

If QI1 is "0" and S1 is "1", this indicates there is some token other than a DATA Token. As is well known in the field of digital electronics, the output of NAND20 will be "1". The NAND gate NAND22 will invert this (as previously explained) and the signal S2 will thus be a "0". As a result, this "0" value will be loaded into latch LO2 at the start of the next PH1 clock phase and the DATA_TOKEN signal will become "0", indicating that the circuitry is not processing a DATA token.

10 If QI1 is "0" and S0 is "0", thereby indicating a DATA token, then the signal S2 will be "1" (regardless of the other input to NAND22 from the output of NAND20). As a result, this "1" value will be loaded into latch LO2 at the start of the next PH1 clock phase and the DATA_TOKEN signal will become "1", indicating that the circuitry is processing a DATA token.

The NOT_DUPLICATE signal (the output signal QO3) is similarly loaded into the latch LI3 on the next rising edge of the clock PH0. The output signal QI3 from the latch LI3 is combined with the output signal QI2 in a gate NAND24 to form the signal S3. As before, Boolean algebra can be used to show that the signal S3 is a "0" only when both of the signals QI2 and QI3 have the value "1". If the signal QI2 becomes a "0", that is, the DATA TOKEN signal is a "0", then the signal S3 becomes a "1". In other words, if there is not a valid DATA TOKEN (QI2 = 0) or the data word is not a duplicate (QI3 = 0), then the signal S3 goes high.

Assume now, that the DATA TOKEN signal remains HIGH for more than one clock signal. Since the NOT_DUPLICATE signal (QO3) is "fed back" to the latch LI3 and will be inverted by the gate NAND 24 (since its other input QI2 is held HIGH), the output signal QO3 will toggle between "0" and "1". If there is no valid DATA Token, however, the signal QI2 will be a "0", and the signal S3 and the output QO3, will be forced

0972658 "043001

The output Q03 (the NOT_DUPLICATE signal) is also fed back and is combined with the output QA1 from the acceptance latch LAIN in a series of logic gates (NAND16 and INV16, which together form an AND gate) that have as their output a "1", only when the signals QA1 and Q03 both have the value "1". As Fig. 8a shows, the output from the AND gate (the gate NAND16 followed by the gate INV16) also forms the acceptance signal, IN_ACCEPT, which is used as described above in the two-wire interface structure.

20 As long as there is a valid DATA_TOKEN (the DATA_TOKEN
signal QO2 is a "1"), the signal QO3 will toggle between the
HIGH and LOW states, so that the input latches will be
enabled and will be able to accept data, at most, during
every other complete cycle of both clock phases PH0, PH1.
25 The additional condition that the following stage be prepared
to accept data, as indicated by a "HIGH" OUT_ACCEPT signal,
must, of course, still be satisfied. The output latch LDOUT
will, therefore, place the same data word onto the output bus
OUT_DATA for at least two full clock cycles. The OUT_VALID
30 signal will be a "1" only when there is both a valid
DATA_TOKEN (QO2 HIGH) and the validation signal QVOUT is
HIGH.

The signal QEIN, which is the extension bit corresponding to MID_DATA, is combined with the signal S3 in a series of

logic gates (INV10 and NAND10) to form a signal S4. During presentation of a DATA Token, each data word MID_DATA will be repeated by loading it into the output latch LDOUT twice. During the first of these, S4 will be forced to a "1" by the
 5 action of NAND10. The signal S4 is loaded in the latch LEOUT to form OUTEXTN at the same time as MID_DATA is loaded into LDOUT to form OUT_DATA[7:0].

Thus, the first time a given MID_DATA is loaded into LEOUT, the associated OUTEXTN will be forced high, whereas,
 10 on the second occasion, OUTEXTN will be the same as the signal QEIN. Now consider the situation during the very last word of a token in which QEIN is known to be low. During the first time MID_DATA is loaded into LDOUT, OUTEXTN will be "1", and during the second time, OUTEXTN will be "0",
 15 indicating the true end of the token.

The output signal QVIN from the validation latch LVIN is combined with the signal QI3 in a similar gate combination (INV12 and NAND12) to form a signal S5. Using known Boolean techniques, it can be shown that the signal S5 is HIGH either
 20 when the validation signal QVIN is HIGH, or when the signal QI3 is low (indicating that the data is a duplicate). The signal S5 is loaded into the validation output latch LVOUT at the same time that MID_DATA is loaded into LDOUT and the intermediate extension bit (signal S4) is loaded into LEOUT.
 25 Signal S5 is also combined with the signal QO2 (the data token signal) in the logic gates NAND30 and INV30 to form the output validation signal OUT_VALID. As was mentioned earlier, OUT_VALID is HIGH only when there is a valid token and the validation signal QVOUT is high.

30 In the present invention, the MID_ACCEPT signal is combined with the signal S5 in a series of logic gates (NAND26 and INV26) that perform the well-known AND function to form a signal S6 that is used as one of the two enabling signals to the latches LO1, LO2 and LO3. The signal S6 rises

to a "1" when the MID_ACCEPT signal is HIGH and when either the validation signal QVIN is high, or when the token is a duplicate (QI3 is a "0"). If the signal MID_ACCEPT is HIGH, the latches LO1-LO3 will, therefore, be enabled when the
 5 clock signal PH1 is high whenever valid input data is loaded at the input of the stage, or when the latched data is a duplicate.

From the discussion above, one can see that the stage shown in Figs. 8a and 8b will receive and transfer data
 10 between stages under the control of the validation and acceptance signals, as in previous embodiments, with the exception that the output signal from the acceptance latch LAIN at the input side is combined with the toggling duplication signal so that a data word will be output twice
 15 before a new word will be accepted.

The various logic gates such as NAND16 and INV16 may, of course, be replaced by equivalent logic circuitry (in this case, a single AND gate). Similarly, if the latches LEIN and LVIN, for example, have inverting outputs, the inverters
 20 INV10 and INV12 will not be necessary. Rather, the corresponding input to the gates NAND10 and NAND12 can be tied directly to the inverting outputs of these latches. As long as the proper logical operation is performed, the stage will operate in the same manner. Data words and extension
 25 bits will still be duplicated.

One should note that the duplication function that the illustrated stage performs will not be performed unless the first data word of the token has a "1" in the third position of the word and "0's" in the five high-order bits. (Of
 30 course, the required pattern can easily be changed and set by selecting other logic gates and interconnections other than the NOR1, NOR2, NND18 gates shown.)

In addition, as Fig. 8 shows, the OUT_VALID signal will be forced low during the entire token unless the first data word

has the structure described above. This has the effect that all tokens except the one that causes the duplication process will be deleted from the token stream, since a device connected to the output terminals (OUTDATA, OUTEXTN and
 5 OUTVALID) will not recognize these token words as valid data.

As before, both validation latches LVIN, LVOUT in the stage can be reset by a single conductor NOT_RESETO, and a single resetting input R on the downstream latch LVOUT, with the reset signal being propagated backwards to cause the
 10 upstream validation latch to be forced low on the next clock cycle.

It should be noted that in the example shown in Fig. 8, the duplication of data contained in DATA tokens serves only as an example of the way in which circuitry may manipulate the ACCEPT and VALID signals so that more data is leaving the
 15 pipeline stage than that which is arriving at the input. Similarly, the example in Fig. 8 removes all non-DATA tokens purely as an illustration of the way in which circuitry may manipulate the VALID signal to remove data from the stream.
 20 In most typical applications, however, a pipeline stage will simply pass on any tokens that it does not recognize, unmodified, so that other stages further down the pipeline may act upon them if required.

Figs. 9a and 9b taken together illustrate an example of a
 25 timing diagram for the data duplication circuit shown in Figs. 8a and 8b. As before, the timing diagram shows the relationship between the two-phase clock signals, the various internal and external control signals, and the manner in which data is clocked between the input and output sides of
 30 the stage and is duplicated.

Referring now more particularly to Figure 10, there is shown a reconfigurable process stage in accordance with one

aspect of the present invention.

Input latches 34 receive an input over a first bus 31. A first output from the input latches 34 is passed over line 32 to a token decode subsystem 33. A second output from the input latches 34 is passed as a first input over line 35 to a processing unit 36. A first output from the token decode subsystem 33 is passed over line 37 as a second input to the processing unit 36. A second output from the token decode 33 is passed over line 40 to an action identification unit 39. The action identification unit 39 also receives input from registers 43 and 44 over line 46. The registers 43 and 44 hold the state of the machine as a whole. This state is determined by the history of tokens previously received. The output from the action identification unit 39 is passed over line 38 as a third input to the processing unit 36. The output from the processing unit 36 is passed to output latches 41. The output from the output latches 41 is passed over a second bus 42.

Referring now to Figure 11, a Start Code Detector (SCD) 51 receives input over a two-wire interface 52. This input can be either in the form of DATA tokens or as data bits in a data stream. A first output from the Start Code Detector 51 is passed over line 53 to a first logical first-in first-out buffer (FIFO) 54. The output from the first FIFO 54 is logically passed over line 55 as a first input to a Huffman decoder 56. A second output from the Start Code Detector 51 is passed over line 57 as a first input to a DRAM interface 58. The DRAM interface 58 also receives input from a buffer manager 59 over line 60. Signals are transmitted to and received from external DRAM (not shown) by the DRAM interface 58 over line 61. A first output from the DRAM interface 58 is passed over line 62 as a first physical input to the Huffman decoder 56.

The output from the Huffman decoder 56 is passed over line 63 as an input to an Index to Data Unit (ITOD) 64. The Huffman decoder 56 and the ITOD 64 work together as a single logical unit. The output from the ITOD 64 is passed over line 65 to an arithmetic logic unit (ALU) 66. A first output from the ALU 66 is passed over line 67 to a read-only memory (ROM) state machine 68. The output from the ROM state machine 68 is passed over line 69 as a second physical input to the Huffman decoder 56. A second output from the ALU 66 is passed over line 70 to a Token Formatter (T/F) 71.

A first output 72 from the T/F 71 of the present invention is passed over line 72 to a second FIFO 73. The output from the second FIFO 73 is passed over line 74 as a first input to an inverse modeller 75. A second output from the T/F 71 is passed over line 76 as a third input to the DRAM interface 58. A third output from the DRAM interface 58 is passed over line 77 as a second input to the inverse modeller 75. The output from the inverse modeller 75 is passed over line 78 as an input to an inverse quantizer 79. The output from the inverse quantizer 79 is passed over line 80 as an input to an inverse zig-zag (IZZ) 81. The output from the IZZ 81 is passed over line 82 as an input to an inverse discrete cosine transform (IDCT) 83. The output from the IDCT 83 is passed over line 84 to a temporal decoder (not shown).

Referring now more particularly to Figure 12, a temporal decoder in accordance with the present invention is shown. A fork 91 receives as input over line 92 the output from the IDCT 83 (shown in Fig. 11). As a first output from the fork 91, the control tokens, e.g., motion vectors and the like, are passed over line 93 to an address generator 94. Data tokens are also passed to the address generator 94 for counting purposes. As a second output from the fork 91, the

data is passed over line 95 to a FIFO 96. The output from the FIFO 96 is then passed over line 97 as a first input to a summer 98. The output from the address generator 94 is passed over line 99 as a first input to a DRAM interface 100.

5 Signals are transmitted to and received from external DRAM (not shown) by the DRAM interface 100 over line 101. A first output from the DRAM interface 100 is passed over line 102 to a prediction filter 103. The output from the prediction filter 103 is passed over line 104 as a second input to the

10 summer 98. A first output from the summer 98 is passed over line 105 to output selector 106. A second output from the summer 98 is passed over line 107 as a second input to the DRAM interface 100. A second output from the DRAM interface 100 is passed over line 108 as a second input to the output

15 selector 106. The output from the output selector 106 is passed over line 109 to a Video Formatter (not shown in Figure 12).

Referring now to Figure 13, a fork 111 receives input from the output selector 106 (shown in Figure 12) over

20 line 112. As a first output from the fork 111, the control tokens are passed over line 113 to an address generator 114. The output from the address generator 114 is passed over line 115 as a first input to a DRAM interface 116. As a second output from the fork 111 the data is passed over line 117 as

25 a second input to the DRAM interface 116. Signals are transmitted to and received from external DRAM (not shown) by the DRAM interface 116 over line 118. The output from the DRAM interface 116 is passed over line 119 to a display pipe 120.

30 It will be apparent from the above descriptions that each line may comprise a plurality of lines, as necessary.

Referring now to Figure 14a, in the MPEG standard a picture 131 is encoded as one or more slices 132. Each slice 132 is, in turn, comprised of a plurality of blocks 133, and is encoded row-by-row, left-to-right in each row. As is shown, each slice 132 may span exactly one full line of blocks 133, less than one line B or D of blocks 133 or multiple lines C of blocks 133.

Referring to Figure 14b, in the JPEG and H.261 standards, the Common Intermediate Format (CIF) is used, wherein a picture 141 is encoded as 6 rows each containing 2 groups of blocks (GOBs) 142. Each GOB 142 is, in turn, composed of either 3 rows or 6 rows of an indeterminate number of blocks 143. Each GOB 142 is encoded in a zigzag direction indicated by the arrow 144. The GOBs 142 are, in turn, processed row-by-row, left-to-right in each row.

Referring now to Figure 14c, it can be seen that, for both MPEG and CIF, the output of the encoder is in the form of a data stream 151. The decoder receives this data stream 151. The decoder can then reconstruct the image according to the format used to encode it. In order to allow the decoder to recognize start and end points for each standard, the data stream 151 is segmented into lengths of 33 blocks 152.

Referring to Figure 15, a Venn diagram is shown, representing the range of values possible for the table selection from the Huffman decoder 56 (shown in Fig. 11) of the present invention. The values possible for an MPEG decoder and an H.261 decoder overlap, indicating that a single table selection will decode both certain MPEG and certain H.261 formats. Likewise, the values possible for an MPEG decoder and a JPEG decoder overlap, indicating that a single table selection will decode both certain MPEG and

certain JPEG formats. Additionally, it is shown that the H.261 values and the JPEG values do not overlap, indicating that no single table selection exists that will decode both formats.

5 Referring now more particularly to Figure 16, there is shown a schematic representation of variable length picture data in accordance with the practice of the present invention. A first picture 161 to be processed contains a first PICTURE_START token 162, first picture information of
10 indeterminate length 163, and a first PICTURE_END token 164. A second picture 165 to be processed contains a second PICTURE_START token 166, second picture information of indeterminate length 167, and a second PICTURE_END token 168. The PICTURE_START tokens 162 and 166 indicate the start of
15 the pictures 161 and 165 to the processor. Likewise, the PICTURE_END tokens 164 and 168 signify the end of the pictures 161 and 165 to the processor. This allows the processor to process picture information 163 and 167 of variable lengths.

20 Referring to Figure 17, a split 171 receives input over line 172. A first output from the split 171 is passed over line 173 to an address generator 174. The address generated by the address generator 174 is passed over line 175 to a DRAM interface 176. Signals are transmitted to and
25 received from external DRAM (not shown) by the DRAM interface 176 over line 177. A first output from the DRAM interface 176 is passed over line 178 to a prediction filter 179. The output from the prediction filter 179 is passed over line 180 as a first input to a summer 181. A second output from the
30 split 171 is passed over line 182 as an input to a first-in first-out buffer (FIFO) 183. The output from the FIFO 183 is passed over line 184 as a second input to the summer 181. The output from the summer 181 is passed over line 185 to a

write signal generator 186. A first output from the write signal generator 186 is passed over line 187 to the DRAM interface 176. A second output from the write signal generator 186 is passed over line 188 as a first input to a read signal generator 189. A second output from the DRAM interface 176 is passed over line 190 as a second input to the read signal generator 189. The output from the read signal generator 189 is passed over line 191 to a Video Formatter (not shown in Figure 17).

Referring now to Figure 18, the prediction filtering process is illustrated. A forward picture 201 is passed over line 202 as a first input to a summer 203. A backward picture 204 is passed over line 205 as a second input to the summer 203. The output from the summer 203 is passed over line 206.

Referring to Figure 19, a slice 211 comprises one or more macroblocks 212. In turn, each macroblock 212 comprises four luminance blocks 213 and two chrominance blocks 214, and contains the information for an original 16 x 16 block of pixels. Each of the four luminance blocks 213 and two chrominance blocks 214 is 8 x 8 pixels in size. The four luminance blocks 213 contain a 1 pixel to 1 pixel mapping of the luminance (Y) information from the original 16 x 16 block of pixels. One chrominance block 214 contains a representation of the chrominance level of the blue color signal (Cu/b), and the other chrominance block 214 contains a representation of the chrominance level of the red color signal (Cv/r). Each chrominance level is subsampled such that each 8 x 8 chrominance block 214 contains the chrominance level of its color signal for the entire original 16 x 16 block of pixels.

Referring now to Figure 20, the structure and function of the Start Code Detector will become apparent. A

value register 221 receives image data over a line 222. The line 222 is eight bits wide, allowing for parallel transmission of eight bits at a time. The output from the value register 221 is passed serially over line 223 to a
5 decode register 224. A first output from the decode register 224 is passed to a detector 225 over a line 226. The line 226 is twenty-four bits wide, allowing for parallel transmission of twenty-four bits at a time. The detector 225 detects the presence or absence of an image which corresponds
10 to a standard-independent start code of 23 "zero" values followed by a single "one" value. An 8-bit data value image follows a valid start code image. On detecting the presence of a start code image, the detector 225 transmits a start image over a line 227 to a value decoder 228.

15 A second output from the decode register 224 is passed serially over line 229 to a value decode shift register 230. The value decode shift register 230 can hold a data value image fifteen bits long. The 8-bit data value following the start code image is shifted to the right of the
20 value decode shift register 230, as indicated by area 231. This process eliminates overlapping start code images, as discussed below. A first output from the value decode shift register 230 is passed to the value decoder 228 over a line 232. The line 232 is fifteen bits wide, allowing for
25 parallel transmission of fifteen bits at a time. The value decoder 228 decodes the value image using a first look-up table (not shown). A second output from the value decode shift register 230 is passed to the value decoder 228 which passes a flag to an index-to-tokens converter 234 over a line
30 235. The value decoder 228 also passes information to the index-to-tokens converter 234 over a line 236. The information is either the data value image or start code index image obtained from the first look-up table. The flag

indicates which form of information is passed. The line 236 is fifteen bits wide, allowing for parallel transmission of fifteen bits at a time. While 15 bits has been chosen here as the width in the present invention it will be appreciated that bits of other lengths may also be used. The index-to-tokens converter 234 converts the information to token images using a second look-up table (not shown) similar to that given in Table 12-3 of the Users Manual. The token images generated by the index-to-tokens converter 234 are then output over a line 237. The line 237 is fifteen bits wide, allowing for parallel transmission of fifteen bits at a time.

Referring to Figure 21, a data stream 241 consisting of individual bits 242 is input to a Start Code Detector (not shown in Figure 21). A first start code image 243 is detected by the Start Code Detector. The Start Code Detector then receives a first data value image 244. Before processing the first data value image 244, the Start Code Detector may detect a second start code image 245, which overlaps the first data value image 244 at a length 246. If this occurs, the Start Code Detector does not process the first data value image 244, and instead receives and processes a second data value image 247.

Referring now to Figure 22, a flag generator 251 receives data as a first input over a line 252. The line 252 is fifteen bits wide, allowing for parallel transmission of fifteen bits at a time. The flag generator 251 also receives a flag as a second input over a line 253, and receives an input valid image over a first two-wire interface 254. A first output from the flag generator 251 is passed over a line 255 to an input valid register (not shown). A second output from the flag generator 251 is passed over a line 256 to a decode index 257. The decode index 257 generates four outputs; a picture start image is passed over a line 258, a

picture number image is passed over a line 259, an insert image is passed over a line 260, and a replace image is passed over a line 261. The data from the flag generator 251 is passed over a line 262a. A header generator 263 uses a look-up table to generate a replace image, which is passed over a line 262b. An extra word generator 264 uses the MPU to generate an insert image, which is passed over a line 262c. Line 262a, and line 262b combine to form a line 262, which is first input to output latches 265. The output latches 265 pass data over a line 266. The line 266 is fifteen bits wide, allowing for parallel transmission of fifteen bits at a time.

The input valid register (not shown) passes an image as a first input to a first OR gate 267 over a line 268. An insert image is passed over a line 269 as a second input to the first OR gate 267. The output from the first OR gate 267 is passed as a first input to a first AND gate 270 over a line 271. The logical negation of a remove image is passed over a line 272 as a second input to the first AND gate 270 is passed as a second input to the output latches 265 over a line 273. The output latches 265 pass an output valid image over a second two-wire interface 274. An output accept image is received over the second two-wire interface 274 by an output accept latch 275. The output from the output accept latch 275 is passed to an output accept register (not shown) over a line 276.

The output accept register (not shown) passes an image as a first input to a second OR gate 277 over a line 278. The logical negation of the output from the input valid register is passed as a second input to the second OR gate 277 over a line 279. The remove image is passed over a line 280 as a third input to the second OR gate 277. The output from the second OR gate 277 is passed as a first input to a second AND gate 281 over a line 282. The logical negation of

5

TABLE 600

| | <u>Format</u> | <u>Image Received</u> | <u>Tokens Generated</u> |
|----|---------------|-----------------------|-------------------------|
| 1. | H.261 | SEQUENCE START | SEQUENCE START |
| | MPEG | PICTURE START | GROUP START |
| 5 | JPEG | (None) | PICTURE START |
| | | | PICTURE DATA |
| 2. | H.261 | (None) | PICTURE END |
| | MPEG | (None) | PADDING |
| | JPEG | (None) | FLUSH |
| 10 | | | STOP AFTER PICTURE |

As set forth in Table 600 which shows a relationship between the absence or presence of standard signals in the certain machine independent control tokens, the detection of an image by the Start Code Detector 51 generates a sequence of machine independent Control Tokens. Each image listed in the "Image Received" column starts the generation of all machine independent control tokens listed in the group in the "Tokens Generated" column. Therefore, as shown in line 1 of Table 600, whenever a "sequence start" image is received during H.261 processing or a "picture start" image is received during MPEG processing, the entire group of four control tokens is generated, each followed by its corresponding data value or values. In addition, as set forth at line 2 of Table 600, the second group of four control tokens is generated at the proper time irrespective of images received by the Start Code Detector 51.

TABLE 601

DISPLAY ORDER: I1 B2 B3 P4 B5 B6 P7 B8 B9 I10
 TRANSMIT ORDER: I1 P4 B2 B3 P7 B5 B6 I10 B8 B9

As shown in line 1 of Table 601 which shows the timing relationship between transmitted pictures and displayed pictures, the picture frames are displayed in numerical order. However, in order to reduce the number of frames that

must be stored in memory, the frames are transmitted in a different order. It is useful to begin the analysis from an intraframe (I frame). The I1 frame is transmitted in the order it is to be displayed. The next predicted frame (P frame), P4, is then transmitted. Then, any bi-directionally interpolated frames (B frames) to be displayed between the I1 frame and P4 frame are transmitted, represented by frames B2 and B3. This allows the transmitted B frames to reference a previous frame (forward prediction) or a future frame (backward prediction). After transmitting all the B frames to be displayed between the I1 frame and the P4 frame, the next P frame, P7, is transmitted. Next, all the B frames to be displayed between the P4 and P7 frames are transmitted, corresponding to B5 and B6. Then, the next I frame, I10, is transmitted. Finally, all the B frames to be displayed between the P7 and I10 frames are transmitted, corresponding to frames B8 and B9. This ordering of transmitted frames requires only two frames to be kept in memory at any one time, and does not require the decoder to wait for the transmission of the next P frame or I frame to display an interjacent B frame.

Further information regarding the structure and operation, as well as the features, objects and advantages, of the invention will become more readily apparent to one of ordinary skill in the art from the ensuing additional detailed description of illustrative embodiment of the invention which, for purposes of clarity and convenience of explanation are grouped and set forth in the following sections:

1. Multi-Standard Configurations
2. JPEG Still Picture Decoding
3. Motion Picture Decompression
4. RAM Memory Map
5. Bitstream Characteristics

6. Reconfigurable Processing Stage
7. Multi-Standard Coding
8. Multi-Standard Processing Circuit-2nd Mode of Operation
9. Start Code Detector
- 5 10. Tokens
11. DRAM Interface
12. Prediction Filter
13. Accessing Registers
14. Microprocessor Interface (MPI)
- 10 15. MPI Read Timing
16. MPI Write Timing
17. Key Hole Address Locations
18. Picture End
19. Flushing Operation
- 15 20. Flush Function
21. Stop-After-Picture
22. Multi-Standard Search Mode
23. Inverse Modeler
24. Inverse Quantizer
- 20 25. Huffman Decoder and Parser
26. Diverse Discrete Cosine Transformer
27. Buffer Manager

1. MULTI-STANDARD CONFIGURATIONS

Since the various compression standards, i.e., JPEG, MPEG and H.261, are well known, as for example as described in the aforementioned United States Patent No. 5,212,742, the detailed specifications of those standards are not repeated here.

As previously mentioned, the present invention is capable of decompressing a variety of differently encoded, picture data bitstreams. In each of the different standards of encoding, some form of output formatter is required to take the data presented at the output of the spatial decoder operating alone, or the serial output of a spatial decoder and temporal decoder operating in combination, (as subsequently described herein in greater detail) and reformatting this output for use, including display in a computer or other display systems, including a video display system. Implementation of this formatting varies significantly between encoding standards and/or the type of display selected.

In a first embodiment, in accordance with the present invention, as previously described with reference to Figures 10-12 an address generator is employed to store a block of formatted data, output from either the first decoder (Spatial Decoder) or the combination of the first decoder (Spatial Decoder) and the second decoder (the Temporal Decoder), and to write the decoded information into and/or from a memory in a raster order. The video formatter described hereinafter provides a wide range of output signal combinations.

In the preferred multi-standard video decoder embodiment of the present invention, the Spatial Decoder and the Temporal Decoder are required to implement both an MPEG encoded signal and an H.261 video decoding system. The DRAM interfaces on both devices are configurable to allow the quantity of DRAM required to be reduced when working with

small picture formats and at low coded data rates. The reconfiguration of these DRAMs will be further described hereinafter with reference to the DRAM interface. Typically, a single 4 megabyte DRAM is required by each of the Temporal Decoder and the Spatial Decoder circuits.

The Spatial Decoder of the present invention performs all the required processing within a single picture. This reduces the redundancy within one picture.

The Temporal Decoder reduces the redundancy between the subject picture with relationship to a picture which arrives prior to the arrival of the subject picture, as well as a picture which arrives after the arrival of the subject picture. One aspect of the Temporal Decoder is to provide an address decode network which handles the complex addressing needs to read out the data associated with all of these pictures with the least number of circuits and with high speed and improved accuracy.

As previously described with reference to Figure 11, the data arrives through the Start Code Detector, a FIFO register which precedes a Huffman decoder and parser, through a second FIFO register, an inverse modeller, an inverse quantizer, inverse zigzag and inverse DCT. The two FIFOs need not be on the chip. In one embodiment, the data does not flow through a FIFO that is on the chip. The data is applied to the DRAM interface, and the FIFO-IN storage register and the FIFO-OUT register is off the chip in both cases. These registers, whose operation is entirely independent of the standards, will subsequently be described herein in further detail.

The majority of the subsystems and stages shown in Figure 11 are actually independent of the particular standard used and include the DRAM interface 58, the buffer manager 59 which is generating addresses for the DRAM interface, the inverse modeller 75, the inverse zig-zag 81 and the inverse

DCT 83. The standard independent units within the Huffman decoder and parser include the ALU 66 and the token formatter 71.

Referring now to Figure 12, the standard-independent units include the DRAM interface 100, the fork 91, the FIFO register 96, the summer 98 and the output selector 106. The standard dependent units are the address generator 94, which is different in H.261 and in MPEG, and the prediction filter 103, which is reconfigurable to have the ability to do both H.261 and MPEG. The JPEG data will flow through the entire machine completely unaltered.

Figure 13 depicts a high level block diagram of the video formatter chip. The vast majority of this chip is independent of the standard. The only items that are affected by the standard is the way the data is written into the DRAM in the case of H.261, which differs from MPEG or JPEG; and that in H.261, it is not necessary to code every single picture. There is some timing information referred to as a temporal reference which provides some information regarding when the pictures are intended to be displayed, and that is also handled by the address generation type of logic in the video formatter.

The remainder of the circuitry embodied in the video formatter, including all of the color space conversion, the up-sampling filters and all of the gamma correction RAMs, is entirely independent of the particular compression standard utilized.

The Start Code Detector of the present invention is dependent on the compression standard in that it has to recognize different start code patterns in the bitstream for each of the standards. For example, H.261 has a 16 bit start code, MPEG has a 24 bit start code and JPEG uses marker codes which are fairly different from the other start codes. Once the Start Code Detector has recognized those different start

codes, its operation is essentially independent of the compression standard. For instance, during searching, apart from the circuitry that recognizes the different category of markers, much of the operation is very similar between the
5 three different compression standards.

The next unit is the state machine 68 (Figure 11) located within the Huffman decoder and parser. Here, the actual circuitry is almost identical for each of the three compression standards. In fact, the only element that is
10 affected by the standard in operation is the reset address of the machine. If just the parser is reset, then it jumps to a different address for each standard. There are, in fact, four standards that are recognized. These standards are H.261, JPEG, MPEG and one other, where the parser enters a
15 piece of code that is used for testing. This illustrates that the circuitry is identical in almost every aspect, but the difference is the program in the microcode for each of the standards. Thus, when operating in H.261, one program is running, and when a different program is running, there is no
20 overlap between them. The same holds true for JPEG, which is a third, completely independent program.

The next unit is the Huffman decoder 56 which functions with the index to data unit 64. Those two units cooperate together to perform the Huffman decoding. Here,
25 the algorithm that is used for Huffman decoding is the same, irrespective of the compression standard. The changes are in which tables are used and whether or not the data coming into the Huffman decoder is inverted. Also, the Huffman decoder itself includes a state machine that understands some aspects
30 of the coding standards. These different operations are selected in response to an instruction coming from the parser state machine. The parser state machine operates with a different program for each of the three compression standards

and issues the correct command to the Huffman decoder at different times consistent with the standard in operation.

The last unit on the chip that is dependent on the compression standard is the inverse quantizer 79, where the mathematics that the inverse quantizer performs are different for each of the different standards. In this regard, a CODING_STANDARD token is decoded and the inverse quantizer 79 remembers which standard it is operating in. Then, any subsequent DATA tokens that happen after that event, but before another CODING_STANDARD may come along, are dealt with in the way indicated by the CODING_STANDARD that has been remembered inside the inverse quantizer. In the detailed description, there is a table illustrating different parameters in the different standards and what circuitry is responding to those different parameters or mathematics.

The address generation, with reference to H.261, differs for each of the subsystems shown in Figure 12 and Figure 13. The address generation in Figure 11, which generates addresses for the two FIFOs before and after the Huffman decoder, does not change depending on the coding standards. Even in H.261, the address generation that happens on that chip is unaltered. Essentially, the difference between these standards is that in MPEG and JPEG, there is an organization of macroblocks that are in linear lines going horizontally across pictures. As best observed in Figure 14a, a first macroblock A covers one full line. A macroblock B covers less than a line. A macroblock C covers multiple lines. The division in MPEG is into slices 132, and a slice may be one horizontal line, A, or it may be part of a horizontal line B, or it may extend from one line into the next line, C. Each of these slices 132 is made up of a row of macroblocks.

In H.261, the organization is rather different because the picture is divided into groups of blocks (GOB).

5

10

15

30

Figure 14c) group of blocks which is one macroblock high. By doing that, exactly the same counting mechanisms used on the Temporal Decoder for counting through the groups of blocks are also used for MPEG.

5 There is a correspondence in the way that the circuitry is designed between an H.261 group of blocks and an MPEG slice. When H.261 data is processed after the Start Code Detector, each group of blocks is preceded by a slice_start_code. The next group of blocks is preceded by
10 the next slice_start code. The counting that goes on inside the Temporal Decoder for counting through this structure pretends that it is a 33 macroblock-long group that is one macroblock high. This is sufficient, although the circuitry also counts every 11th interval. When it counts to the 11th
15 macroblock or the 22nd macroblock, it resets some counters. This is accomplished by simple circuitry with another counter that counts up each macroblock, and when it gets to 11, it resets to zero. The microcode interrogates that and does that work. All the circuitry in the temporal decoder of the
20 present invention is essentially independent of the compression standard with respect to the physical placement of the macroblocks.

In terms of multi-standard adaptability, there are a number of different tables and the circuitry selects the
25 appropriate table for the appropriate standard at the appropriate time. Each standard has multiple tables; the circuitry selects from the set at any given time. Within any one standard, the circuitry selects one table at one time and another table another time. In a different standard, the
30 circuitry selects a different set of tables. There is some intersection between those tables as indicated previously in the discussion of Figure 15. For example, one of the tables used in MPEG is also used in JPEG. The tables are not a completely isolated set. Figure 15 illustrates an H.261

set, an MPEG set and a JPEG set. Note that there is a much greater overlap between the H.261 set and the MPEG set. They are quite common in the tables they utilize. There is a small overlap between MPEG and JPEG, and there is no overlap at all between H.261 and JPEG so that these standards have totally different sets of tables.

As previously indicated, most of the system units are compression standard independent. If a unit is standard independent, and such units need not remember what CODING_STANDARD is being processed. All of the units that are standard dependent remember the compression standard as the CODING_STANDARD token flows by them. When information encoded/decoded in a first coding standard is distributed through the machine, and a machine is changing standards, prior machines under microprocessor control would normally choose to perform in accordance with the H.261 compression standard. The MPU in such prior machines generates signals stating in multiple different places within the machine that the compression standard is changing. The MPU makes changes at different times and, in addition, may flush the pipeline through.

In accordance with the invention, by issuing a change of CODING_STANDARD tokens at the Start Code Detector that is positioned as the first unit in the pipeline, this change of compression standard is readily handled. The token says a certain coding standard is beginning and that control information flows down the machine and configures all the other registers at the appropriate time. The MPU need not program each register.

The prediction token signals how to form predictions using the bits in the bitstream. Depending on which compression standard is operating, the circuitry translates the information that is found in the standard, i.e. from the bitstream into a prediction mode token. This processing is

performed by the Huffman decoder and parser state machine, where it is easy to manipulate bits based on certain conditions. The Start Code Detector generates this prediction mode token. The token then flows down the machine to the circuitry of the Temporal Decoder, which is the device responsible for forming predictions. The circuitry of the spatial decoder interprets the token without having to know what standard it is operating in because the bits in it are invariant in the three different standards. The Spatial Decoder just does what it is told in response to that token. By having these tokens and using them appropriately, the design of other units in the machine is simplified. Although there may be some complications in the program, benefits are received in that some of the hard wired logic which would be difficult to design for multi-standards can be used here.

2. JPEG STILL PICTURE DECODING

As previously indicated, the present invention relates to signal decompression and, more particularly, to the decompression of an encoded video signal, irrespective of the compression standard employed.

One aspect of the present invention is to provide a first decoder circuit (the Spatial Decoder) to decode a first encoded signal (the JPEG encoded video signal) in combination with a second decoder circuit (the Temporal Decoder) to decode a first encoded signal (the MPEG or H.261 encoded video signal) in a pipeline processing system. The Temporal Decoder is not needed for JPEG decoding.

In this regard, the invention facilitates the decompression of a plurality of differently encoded signals through the use of a single pipeline decoder and decompression system. The decoding and decompression pipeline processor is organized on a unique and special configuration which allows the handling of the multi-standard

encoded video signals through the use of techniques all compatible with the single pipeline decoder and processing system. The Spatial Decoder is combined with the Temporal Decoder, and the Video Formatter is used in driving a video display.

Another aspect of the invention is the use of the combination of the Spatial Decoder and the Video Formatter for use with only still pictures. The compression standard independent Spatial Decoder performs all of the data processing within the boundaries of a single picture. Such a decoder handles the spatial decompression of the internal picture data which is passing through the pipeline and is distributed within associated random access memories, standard independent address generation circuits for handling the storage and retrieval of information into the memories. Still picture data is decoded at the output of the Spatial Decoder, and this output is employed as input to the multi-standard, configurable Video Formatter, which then provides an output to the display terminal. In a first sequence of similar pictures, each decompressed picture at the output of the Spatial Decoder is of the same length in bits by the time the picture reaches the output of the Spatial Decoder. A second sequence of pictures may have a totally different picture size and, hence, have a different length when compared to the first length. Again, all such second sequence of similar pictures are of the same length in bits by the time such pictures reach the output of the Spatial Decoder.

Another aspect of the invention is to internally organize the incoming standard dependent bitstream into a sequence of control tokens and DATA tokens, in combination with a plurality of sequentially-positioned reconfigurable processing stages selected and organized to act as a standard-independent, reconfigurable-pipeline-processor.

With regard to JPEG decoding, a single Spatial Decoder with no off chip DRAM can rapidly decode baseline JPEG images. The Spatial Decoder supports all features of baseline JPEG encoding standards. However, the image size
5 that can be decoded may be limited by the size of the output buffer provided. The Spatial Decoder circuit also includes a random access memory circuit, having machine-dependent, standard independent address generation circuits for handling the storage of information into the memories.

10 As previously, indicated the Temporal Decoder is not required to decode JPEG-encoded video. Accordingly, signals carried by DATA tokens pass directly through the Temporal Decoder without further processing when the Temporal Decoder is configured for a JPEG operation.

15 Another aspect of the present invention is to provide in the Spatial Decoder a pair of memory circuits, such as buffer memory circuits, for operating in combination with the Huffman decoder/video demultiplexor circuit (HD & VDM). A first buffer memory is positioned before the HD & VDM, and a
20 second buffer memory is positioned after the HD & VDM. The HD & VDM decodes the bitstream from the binary ones and zeros that are in the standard encoded bitstream and turns such stream into numbers that are used downstream. The advantage of the two buffer system is for implementing a multi-standard
25 decompression system. These two buffers, in combination with the identified implementation of the Huffman decoder, are described hereinafter in greater detail.

A still further aspect of the present multi-standard, decompression circuit is the combination of a Start Code
30 Detector circuit positioned upstream of the first forward buffer operating in combination with the Huffman decoder. One advantage of this combination is increased flexibility in dealing with the input bitstream, particularly padding, which has to be added to the bitstream. The placement of these

identified components, Start Code Detector, memory buffers, and Huffman decoder enhances the handling of certain sequences in the input bitstream.

In addition, off chip DRAMs are used for decoding JPEG-encoded video pictures in real time. The size and speed of the buffers used with the DRAMs will depend on the video encoded data rates.

The coding standards identify all of the standard dependent types of information that is necessary for storage in the DRAMs associated with the Spatial Decoder using standard independent circuitry.

3. MOTION PICTURE DECOMPRESSION

In the present invention, if motion pictures are being decompressed through the steps of decoding, a further Temporal Decoder is necessary. The Temporal Decoder combines the data decoded in the Spatial Decoder with pictures, previously decoded, that are intended for display either before or after the picture being currently decoded. The Temporal Decoder receives, in the picture coded datastream, information to identify this temporally-displaced information. The Temporal Decoder is organized to address temporally and spatially displaced information, retrieve it, and combine it in such a way as to decode the information located in one picture with the picture currently being decoded and ending with a resultant picture that is complete and is suitable for transmission to the video formatter for driving the display screen. Alternatively, the resultant picture can be stored for subsequent use in temporal decoding of subsequent pictures.

Generally, the Temporal Decoder performs the processing between pictures either earlier and/or later in time with reference to the picture currently being decoded. The Temporal Decoder reintroduces information that is not encoded within the coded representation of the picture, because it is

5

10

20

•

30

The Temporal Decoder also reorders the blocks of picture data for display by a display circuit. The address decode circuitry, described hereinafter, provides handling of this reordering.

5 As previously mentioned, one important feature of the Temporal Decoder is to add picture information together from a selection of pictures which have arrived earlier or later than the picture under processing. When a picture is described in this context, it may mean any one of the
10 following:

1. The coded data representation of the picture;
2. The result, i.e., the final decoded picture resulting from the addition of a process step performed by the decoder;
- 15 3. Previously decoded pictures read from the DRAM; and
4. The result of the spatial decoding, i.e., the extent of data between a PICTURE_START token and a subsequent PICTURE_END token.

After the picture data information is processed by the
20 Temporal Decoder, it is either displayed or written back into a picture memory location. This information is then kept for further reference to be used in processing another different coded data picture.

Re-ordering of the MPEG encoded pictures for visual
25 display involves the possibility that a desired scrambled picture can be achieved by varying the re-ordering feature of the Temporal Decoder.

4. RAM MEMORY MAP

The Spatial Decoder, Temporal Decoder and Video
30 Formatter all use external DRAM. Preferably, the same DRAM is used for all three devices. While all three devices use DRAM, and all three devices use a DRAM interface in conjunction with an address generator, what each implements

FORMER 046689

in DRAM is different. That is, each chip, e.g. Spatial Decoder and Temporal Decoder, have a different DRAM interface and address generation circuitry even though they use a similar physical, external DRAM.

5 In brief, the Spatial Decoder implements two FIFOs in the common DRAM. Referring again to Figure 11, one FIFO 54 is positioned before the Huffman decoder 56 and parser, and the other is positioned after the Huffman decoder and parser. The FIFOs are implemented in a relatively straightforward
10 manner. For each FIFO, a particular portion of DRAM is set aside as the physical memory in which the FIFO will be implemented.

 The address generator associated with the Spatial Decoder DRAM interface 58 keeps track of FIFO addresses using
15 two pointers. One pointer points to the first word stored in the FIFO, the other pointer points to the last word stored in the FIFO, thus allowing read/write operation on the appropriate word. When, in the course of a read or write operation, the end of the physical memory is reached, the
20 address generator "wraps around" to the start of the physical memory.

 In brief, the Temporal Decoder of the present invention must be able to store two full pictures or frames of whatever encoding standard (MPEG or H.261) is specified. For
25 simplicity, the physical memory in the DRAM into which the two frames are stored is split into two halves, with each half being dedicated (using appropriate pointers) to a particular one of the two pictures.

 MPEG uses three different picture types: Intra (I),
30 Predicted (P) and Bidirectionally interpolated (B). As previously mentioned, B pictures are based on predictions from two pictures. One picture is from the future and one from the past. I pictures require no further decoding by the Temporal Decoder, but must be stored in one of the two

picture buffers for later use in decoding P and B pictures. Decoding P pictures requires forming predictions from a previously decoded P or I picture. The decoded P picture is stored in a picture buffer for use decoding P and B pictures.

5 B pictures can require predictions from both of the picture buffers. However, B pictures are not stored in the external DRAM.

Note that I and P pictures are not output from the Temporal Decoder as they are decoded. Instead, I and P pictures are written into one of the picture buffers, and are

10 read out only when a subsequent I or P picture arrives for decoding. In other words, the Temporal Decoder relies on subsequent P or I pictures to flush previous pictures out of the two picture buffers, as further discussed hereinafter in the section on flushing. In brief, the Spatial Decoder can

15 provide a fake I or P picture at the end of a video sequence to flush out the last P or I picture. In turn, this fake picture is flushed when a subsequent video sequence starts.

The peak memory band width load occurs when decoding B pictures. The worst case is the B frame may be formed from predictions from both the picture buffers, with all predictions being made to half-pixel accuracy.

20

As previously described, the Temporal Decoder can be configured to provide MPEG picture reordering. With this

25 picture reordering, the output of P and I pictures is delayed until the next P or I picture in the data stream starts to be decoded by the Temporal Decoder.

As the P or I pictures are reordered, certain tokens are stored temporarily on chip as the picture is written into the picture buffers. When the picture is read out for display,

30 these stored tokens are retrieved. At the output of the Temporal Decoder, the DATA Tokens of the newly decoded P or I picture are replaced with DATA Tokens for the older P or I picture.

FOR SECRET

In contrast, H.261 makes predictions only from the picture just decoded. As each picture is decoded, it is written into one of the two picture buffers so it can be used in decoding the next picture. The only DRAM memory operations required are writing 8 x 8 blocks, and forming predictions with integer accuracy motion vectors.

In brief, the Video Formatter stores three frames or pictures. Three pictures need to be stored to accommodate such features as repeating or skipping pictures.

5. BITSTREAM CHARACTERISTICS

Referring now particularly to the Spatial Decoder of the present invention, it is helpful to review the bitstream characteristics of the encoded datastream as these characteristics must be handled by the circuitry of the Spatial Decoder and the Temporal Decoder. For example, under one or more compression standards, the compression ratio of the standard is achieved by varying the number of bits that it uses to code the pictures of a picture. The number of bits can vary by a wide margin. Specifically, this means that the length of a bitstream used to encode a referenced picture of a picture might be identified as being one unit long, another picture might be a number of units long, while still a third picture could be a fraction of that unit.

None of the existing standards (MPEG 1.2, JPEG, H.261) define a way of ending a picture, the implication being that when the next picture starts, the current one has finished. Additionally, the standards (H.261 specifically) allow incomplete pictures to be generated by the encoder.

In accordance with the present invention, there is provided a way of indicating the end of a picture by using one of its tokens: PICTURE_END. The still encoded picture data leaving the Start Code Detector consists of pictures starting with a PICTURE_START token and ending with a

PICTURE_END token, but still of widely varying length. There may be other information transmitted here (between the first and second picture), but it is known that the first picture has finished.

5 The data stream at the output of the Spatial Decoder consists of pictures, still with picture-starts and picture-ends, of the same length (number of bits) for a given sequence. The length of time between a picture-start and a picture-end may vary.

10 The Video Formatter takes these pictures of non-uniform
time and displays them on a screen at a fixed picture rate
determined by the type of display being driven. Different
display rates are used throughout the world, e.g. PAL-NTSC
television standards. This is accomplished by selectively
15 dropping or repeating pictures in a manner which is unique.
Ordinary "frame rate converters," e.g. 2-3 pulldown, operate
with a fixed input picture rate, whereas the Video Formatter
can handle a variable input picture rate.

6. RECONFIGURABLE PROCESSING STAGE

Referring again to Figure 10, the reconfigurable processing stage (RPS) comprises a token decode circuit 33 which is employed to receive the tokens coming from a two wire interface 37 and input latches 34. The output of the token decode circuit 33 is applied to a processing unit 36 over the two-wire interface 37 and an action identification circuit 39. The processing unit 36 is suitable for processing data under the control of the action identification circuit 39. After the processing is completed, the processing unit 36 connects such completed signals to the output, two-wire interface bus 40 through output latches 41.

The action identification decode circuit 39 has an input from the token decode circuit 33 over the two-wire

interface bus 40 and/or from memory circuits 43 and 44 over two-wire interface bus 46. The tokens from the token decode circuit 33 are applied simultaneously to the action identification circuit 39 and the processing unit 36. The
5 action identification function as well as the RPS is described in further detail by tables and figures in a subsequent portion of this specification.

The functional block diagram in Figure 10 illustrates those stages shown in Figures 11, 12 and 13 which
10 are not standard independent circuits. The data flows through the token decode circuit 33, through the processing unit 36 and onto the two-wire interface circuit 42 through the output latches 41. If the Control Token is recognized by the RPS, it is decoded in the token decode circuit 33 and
15 appropriate action will be taken. If it is not recognized, it will be passed unchanged to the output two-wire interface 42 through the output circuit 41. The present invention operates as a pipeline processor having a two-wire interface for controlling the movement of control tokens through the
20 pipeline. This feature of the invention is described in greater detail in the previously filed EPO patent application number 92306038.8.

In the present invention, the token decode circuit 33 is employed for identifying whether the token presently entering
25 through the two-wire interface 42 is a DATA token or control token. In the event that the token being examined by the token decode circuit 33 is recognized, it is exited to the action identification circuit 39 with a proper index signal or flag signal indicating that action is to be taken. At the
30 same time, the token decode circuit 33 provides a proper flag or index signal to the processing unit 36 to alert it to the presence of the token being handled by the action identification circuit 39.

Control tokens may also be processed.

A more detailed description of the various types of tokens usable in the present invention will be subsequently described hereinafter. For the purpose of this portion of the specification, it is sufficient to note that the address carried by the control token is decoded in the decoder 33 and is used to access registers contained within the action identification circuit 39. When the token being examined is a recognized control token, the action identification circuit 39 uses its reconfiguration state circuit for distributing the control signals throughout the state machine. As previously mentioned, this activates the state machine of the action identification decoder 39, which then reconfigures itself. For example, it may change coding standards. In this way, the action identification circuit 39 decodes the required action for handling the particular standard now passing through the state machine shown with reference to Figure 10.

Similarly, the processing unit 36 which is under the control of the action identification circuit 39 is now ready to process the information contained in the data fields of the DATA token when it is appropriate for this to occur. On many occasions, a control token arrives first, reconfigures the action identification circuit 39 and is immediately followed by a DATA token which is then processed by the processing unit 36. The control token exits the output latches circuit 41 over the output two-wire interface 42 immediately preceding the DATA token which has been processed within the processing unit 36.

In the present invention, the action identification circuit, 39, is a state machine holding history state. The registers, 43 and 44 hold information that has been decoded from the token decoder 33 and stored in these registers.

In general, there is shown token decoding and data processing in accordance with the present invention. The data processing is performed as configured by the action identification circuit 39. The action is affected by a number of conditions and is affected by information generally derived from a previously decoded token or, more specifically, information stored from previously decoded tokens in registers 43 and 44, the current token under processing, and the state and history information that the action identification unit 39 has itself acquired. A distinction is thereby shown between Control tokens and DATA tokens.

In any RPS, some tokens are viewed by that RPS unit as being Control tokens in that they affect the operation of the RPS presumably at some subsequent time. Another set of tokens are viewed by the RPS as DATA tokens. Such DATA tokens contain information which is processed by the RPS in a way that is determined by the design of the particular circuitry, the tokens that have been previously decoded and the state of the action identification circuit 39. Although a particular RPS identifies a certain set of tokens for that particular RPS control and another set of tokens as data, that is the view of that particular RPS. Another RPS can have a different view of the same token. Some of the tokens

might be viewed by one RPS unit as DATA Tokens while another RPS unit might decide that it is actually a Control Token. For example, the quantization table information, as far as the Huffman decoder and state machine is concerned, is data, because it arrives on its input as coded data, it gets formatted up into a series of 8 bit words, and they get formed into a token called a quantization table token (QUANT_TABLE) which goes down the processing pipeline. As far as that machine is concerned, all of that was data; it was handling data, transforming one sort of data into another sort of data, which is clearly a function of the processing performed by that portion of the machine. However, when that information gets to the inverse quantizer, it stores the information in that token a plurality of registers. In fact, because there are 64 8-bit numbers and there are many registers, in general, many registers may be present. This information is viewed as control information, and then that control information affects the processing that is done on subsequent DATA tokens because it affects the number that you multiply each data word. There is an example where one stage viewed that token as being data and another stage viewed it as being control.

Token data, in accordance with the invention is almost universally viewed as being data through the machine. One of the important aspects is that, in general, each stage of circuitry that has a token decoder will be looking for a certain set of tokens, and any tokens that it does not recognize will be passed unaltered through the stage and down the pipeline, so that subsequent stages downstream of the current stage have the benefit of seeing those tokens and may respond to them. This is an important feature, namely there can be communication between blocks that are not adjacent to one another using the token mechanism.

Another important feature of the invention is that each of

5

20

30

a constant, K. Another flag tells the inverse quantizer whether to add another constant. The inverse quantizer remembers in a register the CODING_STANDARD token as it flows by the quantizer. When DATA tokens pass thereafter, the
 5 inverse quantizer remembers what the standard is and it looks up the parameters that it needs to apply to the processing elements in order to perform a proper operation. For example, the inverse quantizer will look up whether K is set to 0, or whether it is set to 1 for a particular compression
 10 standard, and will apply that to its processing circuitry.

In a similar sense the Huffman decoder 56 has a number of tables within it, some for JPEG, some for MPEG and some for H.261. The majority of those tables, in fact, will service more than one of those compression standards. Which
 15 tables are used depends on the syntax of the standard. The Huffman decoder works by receiving a command from the state machine which tells it which of the tables to use. Accordingly, the Huffman decoder does not itself directly have a piece of state going into it, which is remembered and
 20 which says what coding it is performing. Rather, it is the combination of the parser state machine and Huffman decoder together that contain information within them.

Regarding the Spatial Decoder of the present invention, the address generation is modified and is similar
 25 to that shown in Figure 10, in that a number of pieces of information are decoded from tokens, such as the coding standard. The coding standard and additional information as well, is recorded in the registers and that affects the progress of the address generator state machine as it steps
 30 through and counts the macroblocks in the system, one after the other. The last stage would be the prediction filter 179 (Figure 17) which operates in one of two modes, either H.261 or MPEG and are easily identified.

7. MULTI-STANDARD CODING

The system of the present invention also provides a combination of the standard-independent indices generation circuits, which are strategically placed throughout the system in combination with the token decode circuits. For example, the system is employed for specifically decoding either the H.261 video standard, or the MPEG video standard or the JPEG video standard. These three compression coding standards specify similar processes to be done on the arriving data, but the structure of the datastreams is different. As previously discussed, it is one of the functions of the Start Code Detector to detect MPEG start-codes, H.261 start-codes, and JPEG marker codes, and convert them all into a form, i.e., a control token which includes a token stream embodying the current coding standard. The control tokens are passed through the pipeline processor, and are used, i.e., decoded, in the state machines to which they are relevant, and are passed through other state machines to which the tokens are not relevant. In this regard, the DATA Tokens are treated in the same fashion, insofar as they are processed only in the state machines that are configurable by the control tokens into processing such DATA Tokens. In the remaining state machines, they pass through unchanged.

More specifically, a control token in accordance with the present invention, can consist of more than one word in the token. In that case, a bit known as the extension bit is set specifying the use of additional words in the token for carrying additional information. Certain of these additional control bits contain indices indicating information for use in corresponding state machines to create a set of standard-independent indices signals. The remaining portions of the token are used to indicate and identify the internal processing control function which is standard for all of the datastreams passing through the pipeline processor. In one

form of the invention, the token extension is used to carry the current coding standard which is decoded by the relative token decode circuits distributed throughout the machine, and is used to reconfigure the action identification circuit 39 of stages throughout the machine wherever it is appropriate to operate under a new coding standard. Additionally, the token decode circuit can indicate whether a control token is related to one of the selected standards which the circuit was designed to handle.

More specifically, an MPEG start code and a JPEG marker are followed by an 8 bit value. The H.261 start code is followed by a 4 bit value. In this context, the Start Code Detector 51, by detecting either an MPEG start-code or a JPEG marker, indicates that the following 8 bits contain the value associated with the start-code. Independently, it can then create a signal which indicates that it is either an MPEG start code or a JPEG marker and not an H.261 start code. In this first instance, the 8 bit value is entered into a decode circuit, part of which creates a signal indicating the index and flag which is used within the current circuit for handling the tokens passing through the circuit. This is also used to insert portions of the control token which will be looked at thereafter to determine which standard is being handled. In this sense, the control token contains a portion indicating that it is related to an MPEG standard, as well as a portion which indicates what type of operation should be performed on the accompanying data. As previously discussed, this information is utilized in the system to reconfigure the processing stage used to perform the function required by the various standards created for that purpose.

For example, with reference to the H.261 start code, it is associated with a 4 bit value which follows immediately after the start code. The Start Code Detector passes this value into the token generator state machine. The value is

applied to an 8 bit decoder which produces a 3 bit start number. The start number is employed to identify the picture-start of a picture number as indicated by the value.

The system also includes a multi-stage parallel processing pipeline operating under the principles of the two-wire interface previously described. Each of the stages comprises a machine generally taking the form illustrated in Figure 10. The token decode circuit 33 is employed to direct the token presently entering the state machine into the action identification circuit 39 or the processing unit 36, as appropriate. The processing unit has been previously reconfigured by the next previous control token into the form needed for handling the current coding standard, which is now entering the processing stage and carried by the next DATA token. Further, in accordance with this aspect of the invention, the succeeding state machines in the processing pipeline can be functioning under one coding standard, i.e., H.261, while a previous stage can be operating under a separate standard, such as MPEG. The same two-wire interface is used for carrying both the control tokens and the DATA Tokens.

The system of the present invention also utilizes control tokens required to decode a number of coding standards with a fixed number of reconfigurable processing stages. More specifically, the PICTURE_END control token is employed because it is important to have an indication of when a picture actually ends. Accordingly, in designing a multi-standard machine, it is necessary to create additional control tokens within the multi-standard pipeline processing machine which will then indicate which one of the standard decoding techniques to use. Such a control token is the PICTURE_END token. This PICTURE_END token is used to indicate that the current picture has finished, to force the buffers to be flushed, and to push the current picture

through the decoder to the display.

8. MULTI-STANDARD PROCESSING CIRCUIT - SECOND MODE OF OPERATION

A compression standard-dependent circuit, in the form of
5 the previously described Start Code Detector, is suitably
interconnected to a compression standard-independent circuit
over an appropriate bus. The standard-dependent circuit is
connected to a combination dependent-independent circuit over
the same bus and an additional bus. The standard-independent
10 circuit applies additional input to the standard dependent-
independent circuit, while the latter provides information
back to the standard-independent circuit. Information from
the standard-independent circuit is applied to the output
over another suitable bus. Table 600 illustrates that the
15 multiple standards applied as the input to the standard-
dependent Start Code Detector 51 include certain bit streams
which have standard-dependent meanings within each encoded
bit stream.

9. START-CODE DETECTOR

20 As previously indicated the Start Code Detector, in
accordance with the present invention, is capable of taking
MPEG, JPEG and H.261 bit streams and generating from them a
sequence of proprietary tokens which are meaningful to the
rest of the decoder. As an example of how multi-standard
25 decoding is achieved, the MPEG (1 and 2) picture_start_code,
the H.261 picture_start_code and the JPEG start_of_scan (SOS)
marker are treated as equivalent by the Start Code Detector,
and all will generate an internal PICTURE_START token. In a
similar way, the MPEG sequence_start_code and the JPEG SOI
30 (start_of_image) marker both generate a machine
sequence_start_token. The H.261 standard, however, has no
equivalent start code. Accordingly, the Start Code Detector,

in response to the first H.261 picture_start_code, will generate a sequence_start token.

None of the above described images are directly used other than in the SCD. Rather, a machine PICTURE_START token, for example, has been deemed to be equivalent to the PICTURE_START images contained in the bit stream. Furthermore, it must be borne in mind that the machine PICTURE_START by itself, is not a direct image of the PICTURE_START in the standard. Rather, it is a control token which is used in combination with other control tokens to provide standard-independent decoding which emulates the operation of the images in each of the compression coding standards. The combination of control tokens in combination with the reconfiguration of circuits, in accordance with the information carried by control tokens, is unique in and of itself, as well as in further combination with indices and/or flags generated by the token decode circuit portion of a respective state machine. A typical reconfigurable state machine will be described subsequently.

Referring again to Table 600, there are shown the names of a group of standard images in the left column. In the right column there are shown the machine dependent control tokens used in the emulation of the standard encoded signal which is present or not used in the standard image.

With reference to Table 600, it can be seen that a machine sequence_start signal is generated by the Start Code Detector, as previously described, when it decodes any one of the standard signals indicated in Table 600. The Start Code Detector creates sequence_start, group_start, sequence_end, slice_start, user-data, extra-data and PICTURE_START tokens for application to the two-wire interface which is used throughout the system. Each of the stages which operate in conjunction with these control tokens are configured by the contents of the tokens, or are configured by indices created

by contents of the tokens, and are prepared to handle data which is expected to be received when the picture DATA Token arrives at that station.

As previously described, one of the compression standards, such as H.261, does not have a sequence_start image in its data stream, nor does it have a PICTURE_END image in its data stream. The Start Code Detector indicates the PICTURE_END point in the incoming bit stream and creates a PICTURE_END token. In this regard, the system of the present invention is intended to carry data words that are fully packed to contain a bit of information in each of the register positions selected for use in the practice of the present invention. To this end, 15 bits have been selected as the number of bits which are passed between two start codes. Of course, it will be appreciated by one of ordinary skill in the art, that a selection can be made to include either greater or fewer than 15 bits. In other words, all 15 bits of a data word being passed from the Start Code Detector into the DRAM interface are required for proper operation. Accordingly, the Start Code Detector creates extra bits, called padding, which it inserts into the last word of a DATA Token. For purposes of illustration 15 data bits has been selected.

To perform the Padding operation, in accordance with the present invention, binary 0 followed by a number of binary 1's are automatically inserted to complete the 15 bit data word. This data is then passed through the coded data buffer and presented to the Huffman decoder, which removes the padding. Thus, an arbitrary number of bits can be passed through a buffer of fixed size and width.

In one embodiment, a slice_start control token is used to identify a slice of the picture. A slice_start control token is employed to segment the picture into smaller regions. The size of the region is chosen by the encoder.

and the Start Code Detector identifies this unique pattern of the slice_start code in order for the machine-dependent state stages, located downstream from the Start Code Detector, to segment the picture being received into smaller regions. The size of the region is chosen by the encoder, recognized by the Start Code Detector and used by the recombination circuitry and control tokens to decompress the encoded picture. The slice_start_codes are principally used for error recovery.

The start codes provide a unique method of starting up the decoder, and this will subsequently be described in further detail. There are a number of advantages in placing the Start Code Detector before the coded data buffer, as opposed to placing the Start Code Detector after the coded data buffer and before the Huffman decoder and video demultiplexor. Locating the Start Code Detector before the first buffer allows it to 1) assemble the tokens, 2) decode the standard control signals, such as start codes, 3) pad the bitstream before the data goes into the buffer, and 4) create the proper sequence of control tokens to empty the buffers, pushing the available data from the buffers into the Huffman Decoder.

Most of the control token output by the Start Code Detector directly reflect syntactic elements of the various picture and video coding standards. The Start Code Detector converts the syntactic elements into control tokens. In addition to these natural tokens, some unique and/or machine-dependent tokens are generated. The unique tokens include those tokens which have been specifically designed for use with the system of the present invention which are unique in and of themselves, and are employed for aiding in the multi-standard nature of the present invention. Examples of such unique tokens include PICTURE_END and CODING_STANDARD.

Tokens are also introduced to remove some of the

FIG. 10

5

15

20

၁၀

standard-dependent coded data with reference to the Start Code Detector.

Each of the standard compression encoding systems employs a unique start code configuration or image which has been selected to identify that particular compression specification. Each of the start codes also carries with it a start code value. The start code value is employed to identify within the language of the standard the type of operation that the start code is associated with. In the multi-standard decoder of the present invention, the compatibility is based upon the control token and DATA token configuration as previously described. Index signals, including flag signals, are circuit-generated within each state machine, and are described hereinafter as appropriate.

The start and/or marker codes contained in the standards, as well as other standard words as opposed to data words, are sometimes identified as images to avoid confusion with the use of code and/or machine-dependent codes to refer to the contents of control and/or DATA tokens used in the machine. Also, the term start code is often used as a generic term to refer to JPEG marker codes as well as MPEG and H.261 start codes. Marker codes and start codes serve the same purpose. Also, the term "flush" is used both to refer to the FLUSH token, and as a verb, for example when referring to flushing the Start Code Detector shift registers (including the signal "flushed"). To avoid confusion, the FLUSH token is always written in upper case. All other uses of the term (verb or noun) are in lower case.

The standard-dependent coded input picture input stream comprises data and start images of varying lengths. The start images carry with them a value telling the user what operation is to be performed on the data which immediately follows according to the standard. However, in the multi-standard pipeline processing system of the present invention,

Every 8 cycles, the 8 bit bytes are loaded into the first shift register. Each byte is loaded into the value shift register 221 (Figure 20), and 8 additional cycles are used to empty it and load the shift register 231. Eight
 5 cycles are used to empty it, so after three of those operations or 24 cycles, there are still three bytes in the 24 bit register. The value decode shift register 230 is still empty.

Assuming that there is now a PICTURE_START word in the
 10 24 bit shift register, the detect cycle recognizes the PICTURE_START code pattern and provides a start signal as its output. Once the detector has detected a start, the byte following it is the value associated with that start code, and this is currently sitting in the value register 221.

15 Since the contents of the detect shift register has been identified as a start code, its contents must be removed from the two wire interface to ensure that no further processing takes place using these 3 bytes. The decode register is emptied, and the value decode shift register 230 waits for
 20 the value to be shifted all the way over to such register.

The contents now of the low order bit positions of the value decode shift register contains a value associated with the PICTURE_START. The Spatial Decoder equivalent to the standard PICTURE_START signal is referred to as the SD
 25 PICTURE_START signal. The SD PICTURE_START signal itself is going to now be contained in the token header, and the value is going to be contained in the extension word to the token header.

10. TOKENS

30 In the practice of the present invention, a token is a universal adaptation unit in the form of an interactive interfacing messenger package for control and/or data functions and is adapted for use with a reconfigurable

processing stage (RPS) which is a stage, which in response to a recognized token, reconfigures itself to perform various operations.

Tokens may be either position dependent or position independent upon the processing stages for performance of various functions. Tokens may also be metamorphic in that they can be altered by a processing stage and then passed down the pipeline for performance of further functions. Tokens may interact with all or less than all of the stages and in this regard may interact with adjacent and/or non-adjacent stages. Tokens may be position dependent for some functions and position independent for other functions, and the specific interaction with a stage may be conditioned by the previous processing history of a stage.

15 A PICTURE_END token is a way of signalling the end of a picture in a multi-standard decoder.

A multi-standard token is a way of mapping MPEG, JPEG and H.261 data streams onto a single decoder using a mixture of standard dependent and standard independent hardware and control tokens.

20 A SEARCH_MODE token is a technique for searching MPEG, JPEG and H.261 data streams which allows random access and enhanced error recovery.

A STOP_AFTER_PICTURE token is a method of achieving a clear end to decoding which signals the end of a picture and clears the decoder pipeline, i.e., channel change.

25 Furthermore, padding a token is a way of passing an arbitrary number of bits through a fixed size, fixed width buffer.

30 The present invention is directed to a pipeline processing system which has a variable configuration which uses tokens and a two-wire system. The use of control tokens and DATA Tokens in combination with a two-wire system facilitates a multi-standard system capable of having

007-0000-01-0001

extended operating capabilities as compared with those systems which do not use control tokens.

The control tokens are generated by circuitry within the decoder processor and emulate the operation of a number of different type standard-dependent signals passing into the serial pipeline processor for handling. The technique used is to study all the parameters of the multi-standards that are selected for processing by the serial processor and noting 1) their similarities, 2) their dissimilarities, 3) their needs and requirements and 4) selecting the correct token function to effectively process all of the standard signals sent into the serial processor. The functions of the tokens are to emulate the standards. A control token function is used partially as an emulation/translation between the standard dependent signals and as an element to transmit control information through the pipeline processor.

In prior art system, a dedicated machine is designed according to well-known techniques to identify the standard and then set up dedicated circuitry by way of microprocessor interfaces. Signals from the microprocessor are used to control the flow of data through the dedicated downstream components. The selection, timing and organization of this decompression function is under the control of fixed logic circuitry as assisted by signals coming from the microprocessor.

In contrast, the system of the present invention configures the downstream functional stages under the control of the control tokens. An option is provided for obtaining needed and/or alternative control from the MPU.

The tokens provide and make a sensible format for communicating information through the decompression circuit pipeline processor. In the design selected hereinafter and used in the preferred embodiment, each word of a token is a minimum of 8 bits wide, and a single token can extend over